CU-PHY-NP 99/03
August 1999

CREIGHTON UNIVERSITY

Department of Physics

2500 California Plaza, Omaha, NE 68178 • (402) 280-2835

System Test for STAR Slow Controls

Jie Lin

Thesis Approved

By

Thomas S. McShane Ed.D. **Major Advisor**

Barbara Brader **Dean**

Abstract

The first commissioning run of the Solenoidal Tracker At RHIC (STAR) project at Brookhaven National Laboratory will take place in June, 1999. Focused on the Quark-Gluon Plasma (QGP) formation and related studies of strongly interacting matter, STAR will observe collisions on an event-by-event basis and search for QGP signatures. A brief introduction to the Relativistic Heavy-Ion Collider (RHIC) physics and corresponding STAR physics, as well as the main STAR hardware structures are presented. The architecture of STAR Slow Controls is described with the introduction of the development tool known as the Experimental Physics and Industrial Control System (EPICS). The author's contribution of integrating and developing the Slow Controls system will be presented.

System Test for STAR Slow Controls

BY

JIE LIN

A THESIS

Submitted to the Faculty of the Graduate School of

Creighton University in Partial Fulfillment of

the Requirements for the Degree of Master

of Science in the Department of Physics

Omaha, Nebraska August, 1999

Acknowledgements

First of all, I would like to take this opportunity to thank the Dean of the Graduate School and the faculty of the Physics Department of Creighton University. It is their generous offer of a full scholarship that brought me to Creigthon and gave me the chance to complete this thesis. Without the support and encouragement from the department faculty, especially my thesis committee members Dr. Robert Kennedy, Dr. Michael Cherney and Fr. Thomas McShane, it would have been impossible for me to make it this far.

Fr. Thomas McShane, who is my thesis advisor and friend, has been with me ever since I started the project. Not only has he been a very good mentor, he also has taught me many of life's philosophies which I would never find in any textbook. I will always enjoy the time that I spent with him in the summer of 1998 at BNL. It was one of the best times of my life. It was not only a learning time, but also a very enjoyable and unforgettable period. I always feel that "I am on vacation, with Fr. McShane..."

My work at Creighton is also firmly guided by another committee member, Dr. Michael Cherney. I enjoy his sense of humor and appreciate the devotion he shows to his work. His incredible motivation has affected me in many ways. It has been a great pleasure to work with him, one of my true friends.

Another Creighton University STAR Slow Controls member, Dr. Jan Chrin who has returned to Europe, also had a very key role in leading me into the EPICS world. We have many common interests, we work in a very similar way and we play soccer on the

same team. The benefits that I got from working with him have shaped the way I think of my future. I was upset when I heard that he would return to Europe, but I sincerely hope that he has a very good life there.

Also, I would like to give special thanks to my closest friend Charles in California. He is the person who always cheers me up with his incredibly energetic personality and he is the person who teaches me the most about America. I must mention all my good friends in Omaha as well. Without their friendship, my life in Omaha would have been very harsh and miserable.

Finally, the best for the last, I would like to thank my family and my girlfriend in China. Even though we are separated by thousands of miles, their love always warms my heart and keeps me standing like a man. My parents have given me their confidence and my girlfriend has given me something to believe in. I owe the whole world to them. If I ever have some success, it will be theirs.

This work was supported in part by the United States Department of Energy under contract number DE-FG03-96ER40991, the Creighton College of Arts and Sciences and the Dean of the Graduate School.

Sometimes, words can be hopelessly inadequate...

TABLE OF CONTENTS

Chapter 3 STAR Slow Controls and the Author's Contribution

Introduction

An introduction to the relativistic heavy-ion physics involved in the Relativistic Heavy-Ion Collider (RHIC) is presented. Some possible ways of detecting the formation of a Quark Gluon Plasma (QGP) are discussed in Chapter 1. An overview of the Solenoidal Tracker At RHIC (STAR) experiment is presented and a most important detector in STAR, the Time Projection Chamber (TPC), is introduced in Chapter 2. The author's contribution to STAR Slow Controls in EPICS is outlined in Chapter 3.

Chapter 1

Relativistic Heavy-Ion Physics at RHIC

1.1 Relativistic Heavy-Ion Physics

Relativistic heavy-ion physics is the study of nucleus-nucleus collisions at high energies with a particle speed close to the speed of light. Under the extreme conditions of high density and temperature, the behavior of nuclear matter is predicted to be different than it is at lower energy. Some new form of matter may exist only under such extreme conditions. Fig. 1.1 shows the phase diagram of nuclear matter which is governed by its equation of state.

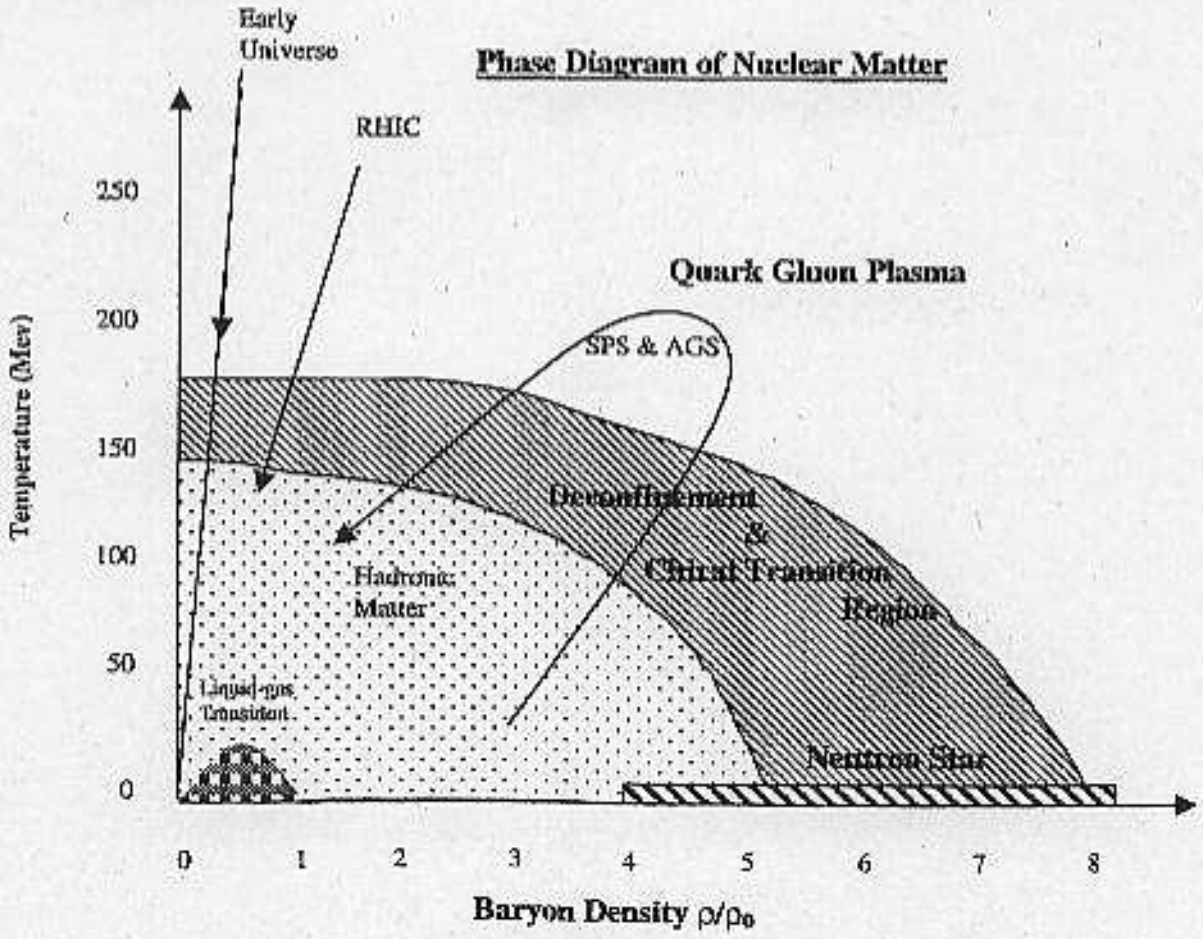

Fig. 1.1: Schematic phase diagram of nuclear matter[1].

Temperature is plotted vs. net baryon-density for an extended volume of nuclear matter in thermal equilibrium. Normal nuclear matter appears at the point shown on the density axis, at about zero temperature: this is the neighborhood studied by traditional nuclear physics experiments. The region of the phase transitions corresponding to quark deconfinement at T_c and chiral symmetry restoration is indicated. Above T_c, hadrons dissolve into quarks and gluons. This region is currently achievable in heavy-ion studies at the Alternating Gradient Synchrotron (AGS) accelerator facility at Brookhaven National Laboratory. Above the temperature of chiral symmetry restoration, the quark's mass decreases because of the shorter distance between quarks. The two critical temperatures may well be coincident. The indicated trajectories show two ways of searching for the Quark Gluon Plasma with high energy nucleus-nucleus collisions: by reaching high baryon-densities among the hot compressed fragments of the colliding nuclei, and at very high temperatures in the central rapidity region among thermally-produced particles where conditions are approximately those of the early universe.

Based on the standard model of particle physics (Quantum ChromoDynamics), the QGP is produced about ten microseconds after the Big Bang, which is believed to be the origin of our universe. QGP may still exist in the cores of very dense stars. In current theories, quarks always interact with other quarks and gluons. Isolated quarks are not expected. The basic constituents of QCD are quarks and anti-quarks interacting through the exchange of color-charged gluons. At short space-time intervals (i.e. large momentum transfers), the effective coupling constant decreases, whereas it becomes strong for large distances and small relative momenta. This results in chiral symmetry breaking and quark-gluon confinement.

At low energy-densities, quarks and gluons are confined in hadrons (baryons and mesons). At the high temperatures anticipated in high-energy collisions of heavy-ions, the baryon-densities and energies are so high that the bonds between quarks and gluons are broken and a deconfined state of quarks and gluons is created, in which quarks and gluons can move around freely. Furthermore, chiral symmetry (left-hand and right-hand terms in the Lagrangian equation) is restored at the critical temperature T_c for baryon-free matter. This novel phase is called the Quark Gluon Plasma (QGP). Even though there always are interactions between quarks and gluons, it is the quark deconfinement that determines the properties of a QGP.

Searching for the formation of a quark-gluon plasma is the major focus of relativistic heavy-ion experiments at higher energies. For this purpose, the Relativistic Heavy-Ion Collider (RHIC) and its associated experiments are currently under construction at Brookhaven National Laboratory, to begin operation in 1999. As seen in the phase diagram, the anticipated temperature and density trajectories at RHIC are expected to lie close to those of the early universe, while those at the AGS occur at higher baryon-densities. In RHIC research, physicists are putting their efforts into recreating this novel state of matter, the Quark-Gluon Plasma, under laboratory conditions with head-on collisions of two gold nuclei with center-of-mass energies of 200 GeV per nucleon.

Relativistic collisions of heavy ions offer a way to study the fundamental theory of strong interactions in the high-density limit and to observe directly the parameters of the predicted phase transition. It may also enable a study of the physical properties of the Quantum ChromoDynamics vacuum state. In such studies, a better understanding of symmetry-breaking mechanisms and the origin of particle masses may be found [1].

1.2 STAR Physics

The STAR (Soleniodal Tracker At RHIC) experiment is currently under construction with operation anticipated at RHIC beginning in June, 1999.

STAR will search for signatures of quark-gluon plasma formation and investigate the behavior of strongly interacting matter at high energy density by focusing on measurements of hadron production over a large solid angle. It utilizes a large volume Time Projection Chamber (TPC) for tracking and particle identification in the high track-density environment provided by RHIC. Besides the TPC, other detectors such as SVT (Silicon Vertex Detector), EMC (ElectroMagnetic Calorimeter), CTB (Central Trigger Barrel), and TOF (Time of Flight), will be included in STAR to measure specific parameters, such as strangeness. STAR will measure many observables simultaneously on an event-by-event basis to study signatures of a possible QGP phase transition and the space-time evolution of the collision process at their respective energies. The goal is to obtain a fundamental understanding of the microscopic structure of hadronic interactions, at the level of quarks and gluons, at high energy densities.

STAR also will study peripheral collisions in which the nuclei physically miss each other, but interact via longer-ranged forces that couple them coherently to the nucleons. The best known example of this is a two-photon collision, studied at the e^+-e^- collider at CERN. But at STAR, photon-pomeron interactions will also be studied.

Table 1.2 describes the STAR physics at RHIC [1].

STAR Physics at RHIC

A. RELATIVISTIC HEAVY-ION COLLISIONS	STAR MEASUREMENTS
(* denotes event-by-event measurement capability)	
Initial Conditions	
Nuclear structure functions/nuclear shadowing	γ jets
(quark and gluon)	jets
Early Gluon-dominated Plasma	
Temperature (radiation)	(Open Charm?), γ's
Quark-Gluon Plasma Transition and Mixed Phase	
Thermalization	$d^3\sigma/dp_t d\eta\, d\phi$ *
Flavor equilibrium	Strangeness ($K^{\pm,0}$, Λ, Ξ, Ω) *
Deconfinment	(J/ψ suppression)
Order of phase transition	T, entropy, HBT (Hanbury Brown-Twiss)
Fluctuations	$d^2E/d\eta\, d\phi$ *, $d^3\sigma/dp_t d\eta\, d\phi$ *
Space-time evolution	$\pi\pi$ *, K $K^{\pm,0}$ HBT
Mini-jet and jet propagation/attenuation	High p_t particles
	γ jets, jets, high p_t ,π^0
Chiral Symmetry Restoration	$\phi \rightarrow$ K $K^{\pm,0}$, (e^+, e^-)
Resonance widths, masses, branching ratios	
Disoriented chiral condensates	E_t (neutral) /n (charged) *
Hadronization Transition	
Freeze-out conditions	HBT* , T*
Strangeness Distillation	K^-, K^+ *
Collision Dynamics	
Stopping and baryochemical potential	$d^3\sigma/dp_t d\eta\, d\phi$ *

Expansion dynamics (long / transverse flow)	Flow, $d^2\sigma/dp_t\,d\eta\,d\phi$ *, $d^2E/d\eta\,d\phi$

B. POLARIZED PROTON-PROTON INTERACTIONS

Spin Physics

Spin-dependent parton distributions	Jet-jet, γ-jet, γ
	$W^{\pm}$, Z^0

C. PERIPHERAL COLLISIONS OF RELATIVISTIC HEAVY-IONS

(PHOTON/POMERON PHYSICS)

Peripheral Collision Physics (γγ, γg, pomerons)

$f_2(1270)$, $f_0(975)$, $\rho^0\rho^0$, $\gamma g \rightarrow c\bar{c}$, pomeron interactions	Resonance decays
	$\pi^+\pi^-$, etc.

Table 1.2 : Summary of the STAR physics program. The table is in time-order as a function of the stage of the collision process being probed. Time starts with the initial states of the incident nuclei at the top and proceed downward. The physics observables are listed on the left and the quantities measured in STAR are listed on the right.

1.2.1 The Quark-Gluon Plasma (QGP)

1.2.1.1 String Theory and the Parton Cascade Model

Six types of quarks have been found. They are known as the up, down, charm, strange, top, and bottom quark which define electric charge and mass (flavor). They are further distinguished by two properties: spin orientation and color charge.

Quarks and anti-quarks are the fundamental components of hadrons. The strong interaction acts only between quarks and gluons. Gluons are the strong force mediators. It is the binding force that holds nuclear constituents together. Hadrons can be grouped into baryons (with three quarks) and mesons (made of quark/anti-quark pairs). Baryons are fermions that obey the Pauli exclusion principle, while mesons are bosons that don't obey the Pauli exclusion principle.

In the very high-energy collisions of nuclei at RHIC, the coherent parton (quark and gluon) wavefunctions of two nuclei evolve into locally quasi-thermal parton distributions, which are the characteristic of the quark-gluon plasma state. There are two main approaches to this evolution: QCD string breaking and partonic cascade.

The string picture, developed from models of soft hadron-hadron interactions, assumes that nuclei pass through each other at collider energies with only a small rapidity [Appendix 1] loss and draws the string (color force) between two nuclei. Not like the Electro-Magnetic interaction in which photons do not interact with each other, gluons will interact with gluons, which will confine the strong interaction to a relatively small region along the beam axis. Fig. 1.2.2.1 shows the schematic interactions between two nuclei with EM interaction and strong interaction.

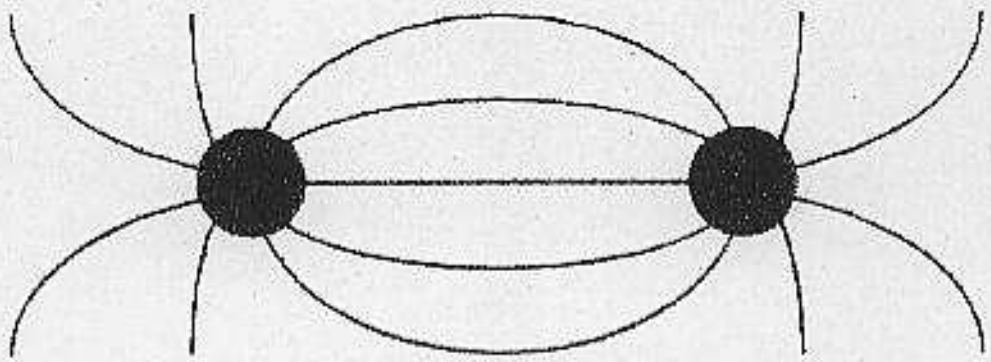

Fig. 1.2.2.1 Confinement and strings [2]

If the area-density of strings is low, they will tend to fragment independently by quark pair production on a proper time scale of about 1fm/c. Most realizations of this picture are based on the Lund String Model[3].

The parton cascade models are designed for the study of non-equilibrium dynamical effects. They are mostly applied to the initial compressional phase and the high-density phase of ultra-relativistic heavy-ion collisions. The parton cascade models all contain this general structure: initialization, when the nucleons of the colliding nuclei are resolved into their parton constituents and yield the initial parton distribution; interaction, when the multiple scatterings occur together with associated parton emission; and hadronization, when partons are recombined via string fragments into final hadron states.

One of the central questions addressed by parton cascades involves the energy deposition processes in space-time as well as momentum space. Partonic cascades predict

that roughly 50% of the expected energy deposition at RHIC takes place at the partonic level [3]. Rapid thermalization is caused by radiative energy degradation and spatial separation of partons with widely different rapidities; transverse momentum (component of momentum perpendicular to the beam axis) distributions of initially scattered partons are almost exponential if radiative corrections are taken into account [4]. For RHIC energies, thermalization is predicted on a proper time scale of 0.3-0.5 fm/c which is on the order of 10^{-24} seconds [5].

Whereas the string picture runs into conceptual difficulties at very high energy when the string densities become too large, the parton cascade model becomes invalid at lower energies, where most partonic scatterings are too soft to be described by perturbative QCD [3].

1.2.1.2 Formation and Evolution of the QGP

Most high-energy models view a heavy-ion collision at RHIC as consisting of three distinct stages, namely the initial Lorenz-contracted hard parton-parton scattering, then a pre-equilibrium or quasi-thermal stage, and finally the hadronic equilibration as the result of rescattering of these partons.

Different physics governs evolution in these three states. The first stage is well-described by perturbative QCD if the momentum transfer involved is sufficiently large. The second stage is addressed by parton cascade models [6,7]. The third stage, which includes the hadronization and subsequent final-state interactions, has recently been modeled by combining an initial parton cascade with either hydrodynamical or string fragmentation-like transport codes. This merging of the three stages of the interaction into one continuous process seems the most complete way of describing the RHIC interactions [8].

This evolution in space-time is shown in Fig. 1.2.1 .

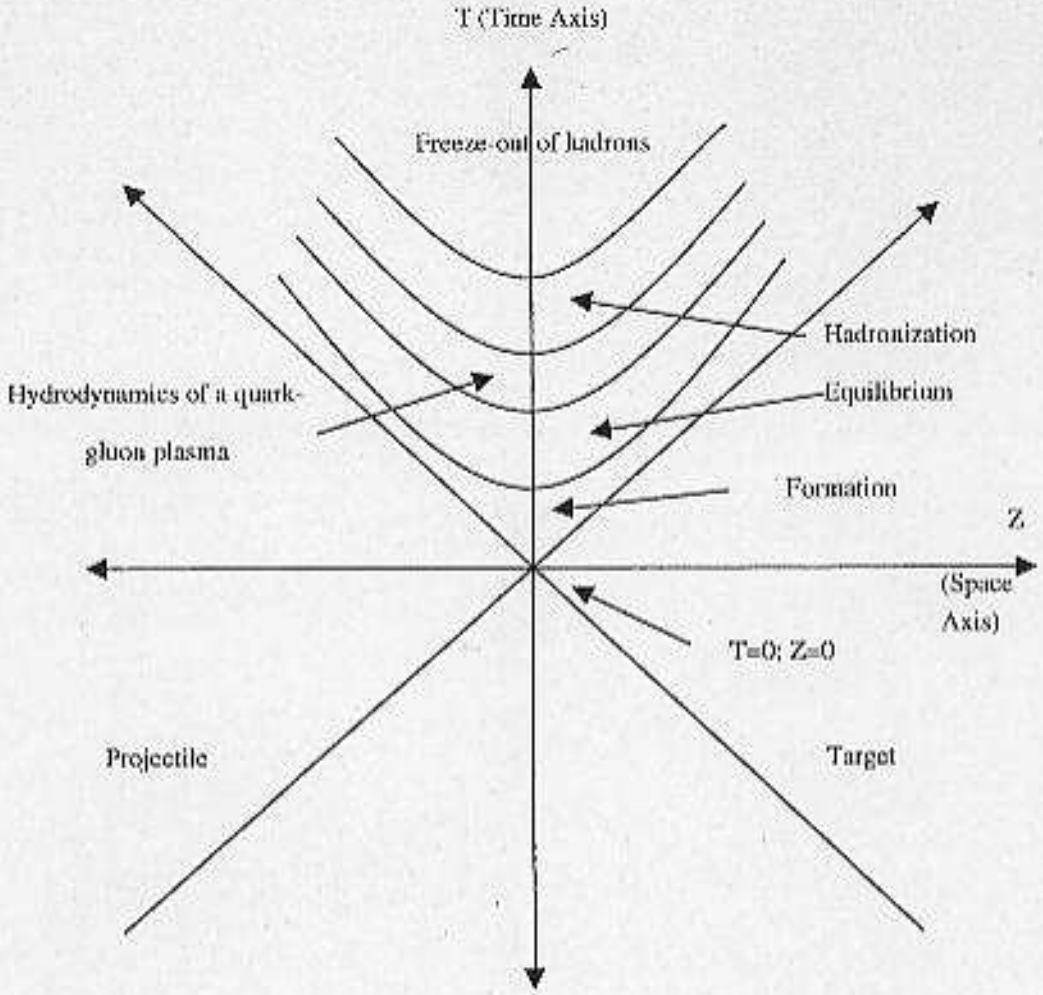

Fig. 1.2.1 Evolution of a QGP in space-time [9]

As indicated in Fig. 1.2.1, the scattered partons develop an incoherent identity immediately after the collision and evolve into a quasi-thermal phase space distribution by a free propagation along the longitudinal direction.

Once the quark-gluon plasma has reached local thermal equilibrium, its further evolution can be described in a framework of relativistic hydrodynamics [10]. But thermal equilibrium does not always imply chemical equilibrium at the parton level. So

the chemical equilibrium of heavier quarks, especially strange and charmed quarks, is thought to be an excellent probe of the physical conditions in dense hadronic matter. A general result of all partonic cascade simulations is that phase-space equilibrium occurs much faster for gluons than for quarks. As a result, the QCD plasma in its hottest stage is predominantly a gluon plasma [11].

The quark-gluon plasma then evolves into its hadronization phase. The plasma expands and cools until it reaches the critical temperature $T_c \approx 170$ MeV and then converts into a hadronic gas while maintaining thermal and chemical equilibrium.

It is difficult to find a robust theoretical description of relativistic heavy-ion collisions involving the QCD phase transition that will predict values for observables. Another major unknown is the influence of the non-equilibrium evolution on the (small) many-body system. So, a strategy for detection of quark matter must collect at least indirect evidence from several "signals" [12]. In the following sections, several of the possible QGP signatures will be discussed.

1.2.2 Quark-Gluon Plasma Signatures

Theory remains academic unless experimental tools verify it. Because there are formidable background signals from the hot hadronic gas phase that follows the hadronization of the plasma phase, it is very difficult to distinguish the QGP signal from the background signal. Much effort is currently devoted to recognizing appropriate signatures for the QGP.

Most of these signatures can be categorized in the following way: (i) signals sensitive to the equation of state; (ii) signals of chiral symmetry restoration; (iii) probes of the color response function (including deconfinement); (iv) probes of the electromagnetic response function; (v) various other signals that escape simple classification [5].

1.2.2.1 Probes of the Equation of State --- thermodynamic variables, nuclear stopping power and collective flow

The basic idea behind this class of signatures is the identification of modifications in the dependence of energy-density ε, pressure p, and entropy-density s of the interacting system as a function of the temperature T and the baryochemical potential μ_B. If a phase transition to QGP occurs, a rapid rise in the effective number of degrees of freedom, expressed by ε/T^4 or P/T^4, should be observed over a small range of T as indicated in Fig. 1.2.2.1.

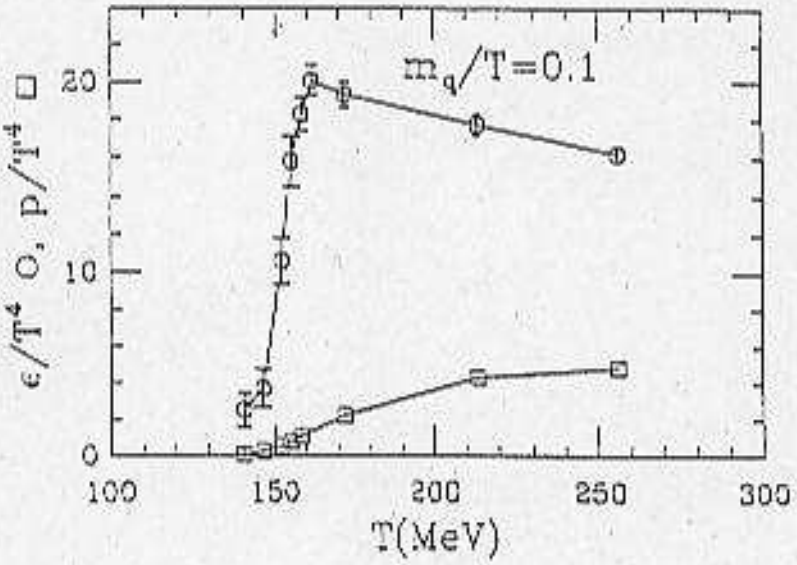

Fig. 1.2.2.1: Results of a lattice QCD calculation using two light quarks flavor [13]. Ploted are the energy-density ε/T^4 (circles) and the pressure P/T^4 (squares) as a function of the temperature T. A transition occurs in the calculation at T=160 MeV.

The nuclear stopping power must be mentioned at this point because it is associated with the thermodynamics signal, energy-density or pressure for example, and several different signatures. The term "nuclear stopping power" characterizes the degree of stopping which an incident nucleon undergoes when it collides with another nucleus. It

determines how much energy is left in the interaction region and how much transverse energy or transverse momentum has escaped from the collision area.

The degree of stopping can be used to estimate the transverse momentum p_t or the shape of the baryon-rapidity distribution. According to the definition of rapidity [Appendix 1], "mid-rapidity region" refers to a situation in which p_z (particle momentum along the beam axis) approaches zero in the center-of-mass frame and the momentum transfer to the transverse direction is maximized. So a rapid change in the shape of the scaled baryon-rapidity distribution with increasing temperature will indicate new degrees of freedom.

The degree of stopping is also particularly interesting for estimation of the energy-density in the interaction region within the Bjorken scenario of scaling hydrodynamics [10]. For such an estimation, the rapidity distribution of the secondary particles is required, not that of the incident particles.

"Flow" is another term that is often used to describe the transverse collective radial and direct energy or momentum flow and is the earliest predicted signature for probing compressed nuclear matter. It has been shown that the excitation function of flow is sensitive to the equation of state and can be used for abnormal matter states and phase transitions [14].

In the fluid dynamical approach, the transverse collective flow is indirectly linked to the pressure of the matter in the reaction zone, shown by the formula below [15],

$$p_X = \int_t \int_A P(\rho, S)\, dA\, dt$$

where dA represents the surface element between the participants (particles that participate in the interaction) and spectators (particles that do not participate the interaction) and the total pressure is the sum of the potential pressure (pressure caused by the potential energy between particles) and the kinetic pressure (pressure caused by the

kinetic energy of the particles). This equation shows that the transverse collective flow depends directly on the equation of state, P (ρ, S).

1.2.2.2 Strangeness Enhancement

Strangeness enhancement is one of the signatures indicating chiral symmetry restoration. It is the most-often proposed signature for a restoration of spontaneously broken chiral symmetry in dense baryon-rich hadronic matter.

As mentioned in section 1.2.1.2, the QCD plasma in its hottest stage, or the initial hard scattering stage, is predominantly a gluon plasma. Therefore, the main contribution of the strangeness formation in the early part of the collision is due to gluon fusion, gluon decay and gluon scattering. This is where the strangeness enhancement is introduced. The energy threshold for producing a strange/anti-strange pair is much lower in a QGP than in a hadron gas. This is because the strange quarks can be produced on their own from gluon interactions in a QGP; it is not necessary to provide also the up and down quarks needed to form a color-neutral hadron as is the case in a hadron gas.

Early work by Geiger [16] predicted a very large strangeness enhancement due to a phase transition. The strange quarks will most likely not chemically equilibrate [17], but the parton cascade might lead to fast local thermal equilibration of the strange quarks. Thus, there is a significant strangeness formation in the parton cascade, mostly due to the re-interaction of soft (low energy) gluons.

At RHIC, kaons will account for almost 90% of the strangeness yield, based on standard string fragmentation models. Strange baryons, though very interesting as rare probes of strangeness equilibration, will contribute only moderately to the strangeness yield [8].

In many theoretical predictions for strange quark matter formation in RHIC, the main requirements are large baryon-densities and a relatively small bag pressure (according to the MIT bag Model in which baryons are all confined inside the bag with little interactions between each other) and energy-density. Based on these arguments, the probability of forming a strangelet in the central rapidity region at RHIC seems low, but the fact that forward rapidity regions at RHIC are expected to have high net baryon-densities might lead to an interesting phenomenon, the possible formation of two phases of QGP separated in rapidity, distinguishable by their respective baryonic content. They have yet to be demonstrated by experiment.

1.2.2.3 Quarkonium Suppression — J/ψ Suppression

The J/ψ meson contains of a $c\bar{c}$ quark pair. In a QGP, the prediction is that the formation of J/ψ mesons is suppressed. This is a result of the Debye screening of a charm/anti-charm quark pair ($c\bar{c}$), initially formed in the QGP by fusion of two incident gluons. Debye screening occurs as charges (or colors) are surrounded by moving charges of the opposite sign (or complementary color), substantially reducing the effective range of the strong force. "Screening length" is defined as the distance between two nuclei at which the binding force begins to collapse. Thus, a bound state of a ($c\bar{c}$) pair cannot exist when the color screening length λ_D is less than the bound state radius. Therefore, a $c\bar{c}$ pair produced in the fusion of two gluons from the colliding nuclei cannot bind inside the quark-gluon plasma and will tend to separate [18]. In addition, the D-meson is expected to dissociate in the deconfined phase, lowering the energy threshold for thermal break-up of the J/ψ into two D-mesons. The combination of these two effects gives rise to a rapid

rise of the dissociation probability beyond the critical temperature T_c, approaching unity at about 1.2 T_c [19]. Less tightly-bound excited states of the ($c\bar{c}$) system, such as ψ' and χ_{c1}, are more easily separated, will be suppressed even more than the J/ψ, and should disappear as soon as the temperature exceeds T_c.

The NA50 Collaboration at CERN has measured the production of J/ψ and ψ' in 158-GeV/c-per-nucleon Pb+Pb reactions using a muon-pair spectrometer [1]. They observed an anomalous suppression of the J/ψ and ψ' normalized to Drell-Yan pair production, a process that describes the annihilation of a quark of one hadron with an anti-quark of another hadron. Fig. 1.2.2.3 shows this suppression compared to the suppression already measured in lighter systems with proton-, oxygen- and sulfur-nucleus reactions in NA38. The ratio of $\sigma^{J/\psi}/\sigma^{DY}$ times the muon pair's branching ratio $B_{\mu\mu}$ is plotted in the figure as a function of L in fermi, where L is defined by the NA50 collaboration as a geometrical mean path length of the ($c\bar{c}$) system.

The proton-, oxygen- (not shown) and sulfur-nucleus reactions are well reproduced by the straight-line fit in the figure, which represents $B_{\mu\mu}\,\sigma^{J/\psi}/\sigma^{DY} =$ $\exp(-\rho\sigma_{abs}L)$, with σ_{abs}=6.2±0.7 mb for the normal absorption of a $c\bar{c}$ state. Most of the peripheral Pb+Pb data agree with the normal absorption model. However, for L>7.5 fm, the Pb+Pb data are strongly suppressed by factors as large as 0.62±0.04.

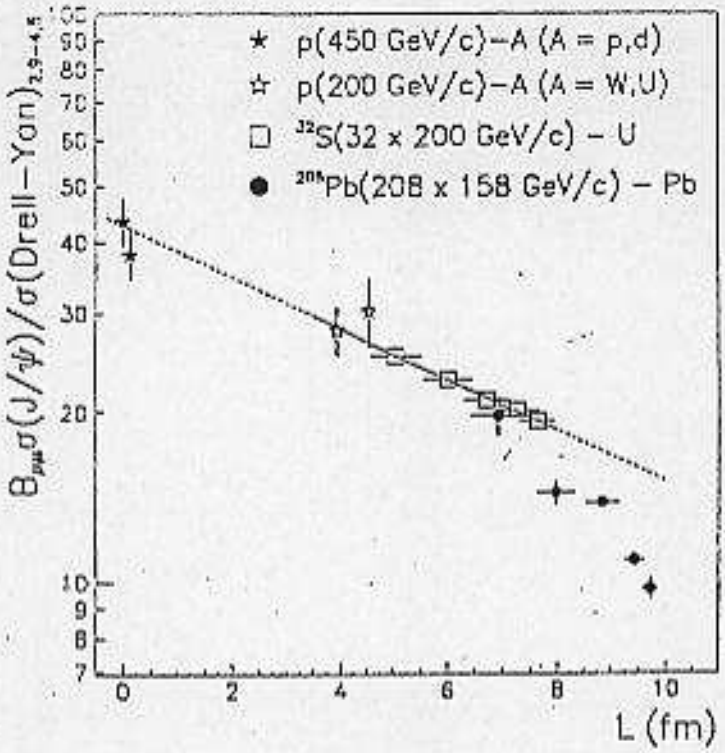

Fig. 1.2.2.3: The ratio $B_{\mu\mu}\ \sigma^{J/\psi}/\sigma^{DY}$ plotted as a function of the geometrical mean path length L(b) of the $c\bar{c}$ system traveling through nuclear matter for the various colliding nuclear systems [20].

Data which have been reported recently by NA50 [20], support this observation with greater significance.

Due to its finite size, the formation of a $(c\bar{c})$ bound state requires a time of order of 1 fm/c. Classically, this can be viewed as the time required for the $(c\bar{c})$ pair to become separated by a distance equal to the average bound state radii. The J/ψ may still survive if the quark-gluon plasma cools very fast or if the J/ψ has high enough transverse

momentum p_t, under which circumstances the J/ψ escapes from the high density and temperature region before the $(c\bar{c})$ pair is separated by more than the bound-state radius.

On the other hand, the J/ψ may also be destroyed in hadron interactions by sufficiently energetic collisions with hadronic co-movers (pions and resonances produced in the same rapidity range as the $c\bar{c}$ pair), leading to a break-up into two D-mesons.

It is still a matter of dispute whether these mechanisms explain the full extent of observed J/ψ suppressions. One of the main problems in the interpretation of the observed suppression as a signal for deconfinement is that non-equilibrium dynamical sources of charmonium suppression have also been clearly discovered in p+A reactions. But in p+A reactions, the formation of an equilibrated QGP is not expected. So it is still not clear that the observation of charmonium suppression definitely leads to the formation of a QGP. Furthermore, the hadronic co-movers and a quark-gluon plasma can describe the available data equally well, at least at low energies [21]. In view of this persistent ambiguity, the observed J/ψ suppression cannot yet be considered as strong evidence for the formation of a quark-gluon plasma in nuclear collisions [5]. Whether this uncertainty can be resolved at higher energies remains to be seen.

Despite prevailing ambiguities, heavy vector-meson suppression remains a good indicator of the presence of a high-density environment in the central rapidity region formed in relativistic heavy-ion collisions.

1.2.2.4 High p_t probes of QCD

Measurements of the fragmentation products of hard scattering can provide information on the matter through which the hard-scattered parton propagates. Thus, another possible way of probing the color structure of QCD matter is to study its effects on the propagation of a fast parton. The mechanisms are similar to those responsible for the electromagnetic energy loss of a fast charged particle in matter: energy may be lost either by excitation of the penetrated medium or by radiation.

The connection between energy loss of a quark and the color-dielectric polarizability of the medium can be established in analogy with the theory of electromagnetic energy loss [22]. The magnitude of the energy loss is proportional to the strong coupling constant based on the stopping power. As mentioned earlier, the greater the nuclear stopping power, the more energy that gets lost in the reaction region. Different authors obtain a stopping power between 0.4 and 1 GeV/fm for a fast quark [22]. This is somewhat smaller than the energy loss of a fast quark in nuclear matter.

Although radiation is a very efficient energy-loss mechanism for relativistic particles, it is strongly suppressed in a dense medium by the Landau-Pomeranchuk effect [23]. The suppression of soft radiation [24] limits the radiative energy loss of a hard parton to about 1 GeV/fm [25]. Adding the two contributions, the stopping power of a quark-gluon plasma is predicted to be higher than that of hadronic matter. In other words, a higher transverse momentum will be observed in events associated with a quark-gluon plasma.

1.2.2.5 Direct photons and thermal dileptons

Photons as well as leptons may be produced from a quark/anti-quark pair in a QGP. They will not interact via the strong force. Thus, they are, in many respects, the cleanest signal

for the quark-gluon plasma because they probe the earliest and hottest phase of the evolution of the thermal fireball, and escape the interaction region without subsequent interaction or modification due to final-state interactions. Their drawbacks are the rather small yields and the relatively large background from hadronic decay processes, especially electromagnetic decays of hadrons, such as π^0, η decay and lepton pair production from π annihilation.

Lepton pairs also provide an early probe for the formation of a QGP. Analogous to the formation of a real photon via a quark/anti-quark annihilation, a virtual photon may be created in the same manner; it subsequently decays into a dilepton. Bremsstrahlung from quarks scattering off gluons can also result in dileptons. They can carry information on the thermodynamic state of the medium at the moment of production in the very same manner as the direct photons.

Many of the original calculations concentrated on lepton pairs emitted with invariant masses in the energy range below the ρ-meson mass (about 770 MeV). However, recent work has shown that lepton pairs from the QGP are also expected to be observed for invariant masses above 1 GeV [26]. Assuming thermalization of the QGP on a time scale of about 1 fm/c, the thermal dilepton spectrum is superseded by Drell-Yan pairs from nucleon-nucleon collisions for dilepton invariant masses around 2-2.5 GeV. There is an indication from the parton cascade model [27] that lepton pairs from a QGP may dominate over Drell-Yan pairs for even higher masses in the 5-10 GeV region. Thus, the early thermal evolution of the quark-gluon phase can be traced in a rather model-independent way [28]. Dileptons from charm decay are predicted to yield a substantial contribution to the total dilepton spectrum and could, because of their different

kinematics, provide a useful determination of the total charm yield [29]. This could serve as an indirect probe of the partonic pre-equilibrium phase, where the total charm yield is enhanced due to rescattering of gluons [30].

Direct photons, the second type of electromagnetic probes of dense matter, compete with a formidable background from the other decay processes. Even if the decay photons can be subtracted, there remains the competition between radiation from the QGP and from the hadronic phase. The most prominent process for the creation of direct (thermal) photons in a QGP are $q\bar{q} \rightarrow \gamma g$ (annihilation) and $gq \rightarrow \gamma q$ (Compton scattering). The production rate and the momentum distribution of the photons depend on the momentum distributions of quarks, anti-quarks and gluons in the plasma. They also carry information on the thermodynamic state of the medium at the moment of production. The main hadronic background processes are pion annihilation $\pi\pi \rightarrow \gamma\rho$ and Compton scattering $\pi\rho \rightarrow \gamma\pi$ [31].

1.2.2.6 Summary

The quark-gluon plasma has yet to be observed unambiguously, although there have been several interesting properties of high-density nuclear matter observed in AGS and SPS experiments. This suggests that studies using relativistic heavy ions at significantly higher energies at RHIC and the LHC will be productive.

Chapter 2

Relativistic Heavy-Ion Collider (RHIC) and Experiments

2.1 The RHIC Facilities

The Relativistic Heavy-Ion Collider is presently under construction and expected to begin operation in June, 1999. A schematic diagram of the RHIC accelerator complex at Brookhaven is displayed in Fig. 2.1.1.

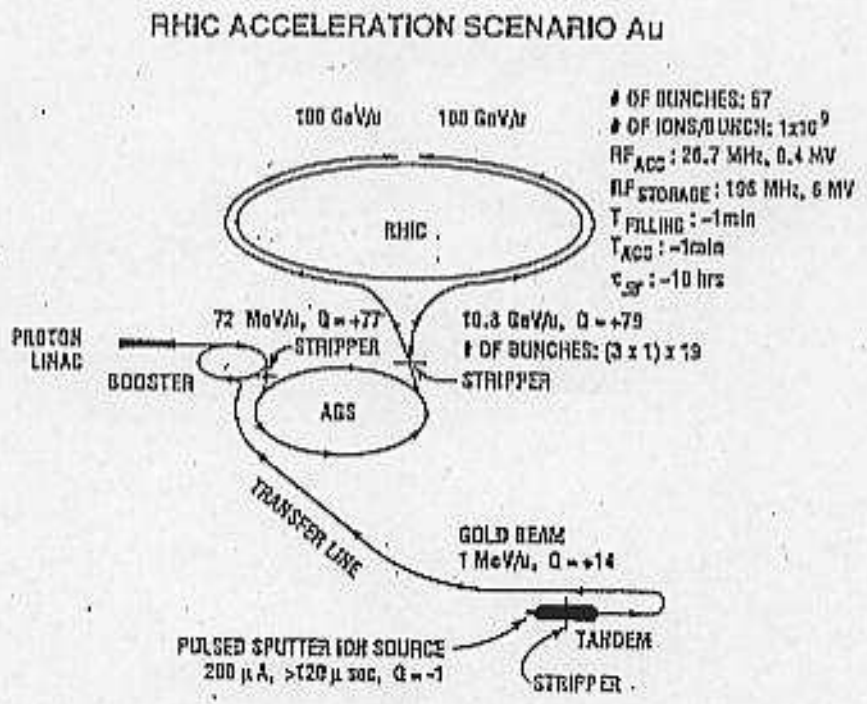

Fig. 2.1.1: The Relativistic Heavy-Ion Collider (RHIC) accelerator complex at Brookhaven National Laboratory. Details of the characteristics of proton and Au beams are also indicated after acceleration in each phase.

Ion beams are accelerated from the Tandem Van de Graaff, through the transfer line into the AGS Booster and AGS prior to injection into RHIC. RHIC will accelerate and collide ions from protons up to the heaviest nuclei over a range of energies, up to 250 GeV for protons and 100 GeV/nucleon for Au nuclei.

Approximately 1000 charged particles per unit of pseudorapidity will be produced from the collisions of the heaviest nuclei at impact parameters near zero at RHIC. This makes it very difficult to detect all of the products from reactions. The experiments will take various approaches to search for the QGP. Two large collider detectors, STAR and PHENIX, are under construction for operation at RHIC start-up. The STAR experiment will concentrate on event-by-event measurements of hadron production over a large solid angle in order to study global observables, such as fluctuations of particle ratios, energy-density, and entropy-density.

There are two other smaller detectors in the RHIC ring. They are BRAHMS, a forward and mid-rapidity hadron spectrometer, and PHOBOS, a compact multi-particle spectrometer. The collaborations constructing these detector systems consist of approximately 900 scientists from over 80 institutions.

2.2 The Solenoidal Tracker At RHIC (STAR) Experiment

The STAR (Solenoidal Tracker At RHIC) experiment is presently under construction with operation expected at RHIC beginning in June, 1999.

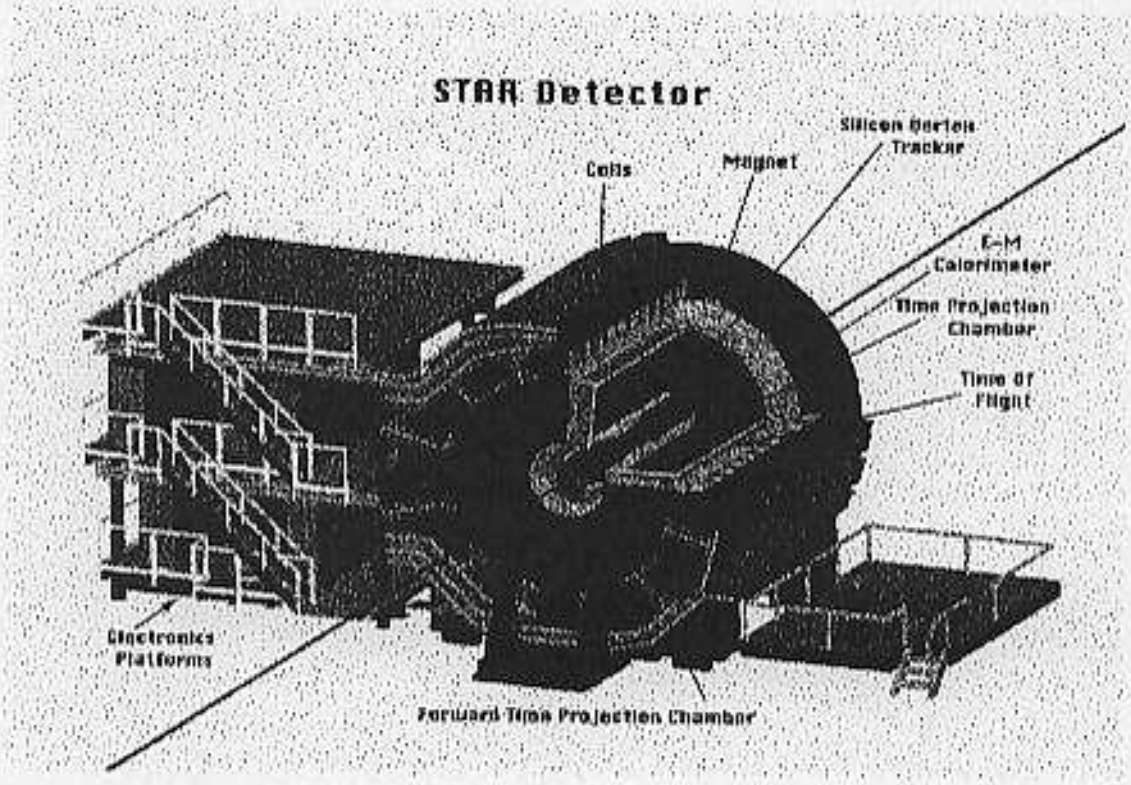

Fig. 2.2: Layout of the STAR Detector

STAR is a close-to-2π coverage detector consisting of a central solenoidal magnet which contains the central Time Projection Chamber (TPC), the main detector; a Silicon Vertex Tracker (SVT), the detector able to locate the decays of strange particles; plus an Electro-Magnetic Calorimeter (EMC), the transverse energy detector. The solenoid provides a uniform magnet field of 0.5 T for tracking, momentum analysis and particle identification in the tracking detector. Along the beam line the coverage is extended by two Forward Time Projection Chambers (FTPC), one on each side.

The following trigger detectors are under construction for STAR: a central trigger barrel for determinging the charged-particle multiplicity surrounding the outer cylinder of the TPC, will trigger in the $|\eta|<1$ region; a TPC end-cap trigger will be triggering on charged-particle multiplicity over the interval $1<|\eta|<2$; a vertex-position detector will be used to identify and localize the interaction vertex; and a zero-degree calorimeter (at large pseudorapidity, not shown in Fig. 2.2) will be exploited to avoid large impact-parameter collisions that have a large amount of energy remaining along the beam line.

2.3 The Time Projection Chamber (TPC)

The STAR Time Projection Chamber (TPC) is the main detector of the STAR experiment. It is designed to track the high-energy particles in each collision event. Its length is 4.18 meters between the two end caps with a total outer diameter of 4.1 meters. The outer field cage contains a gas mixture of 90% Argon and 10% Methane, called P-10. The gas acts as an ionization medium as the charged particles travel through its volume. The average electric field applied between the central membrane and the two end caps is 145 V/cm and a 0.5-T magnetic field is present inside the TPC. The magnetic field causes the ionizing particle to be detected. The electric fields force the ionized electrons to drift towards the end caps of the TPC. Energy and momentum measurements are made from these curved ionization tracks that are left in the TPC gas.

Fig. 2.3 shows a schematic diagram of major TPC components. The TPC gas system and field cage will be briefly described.

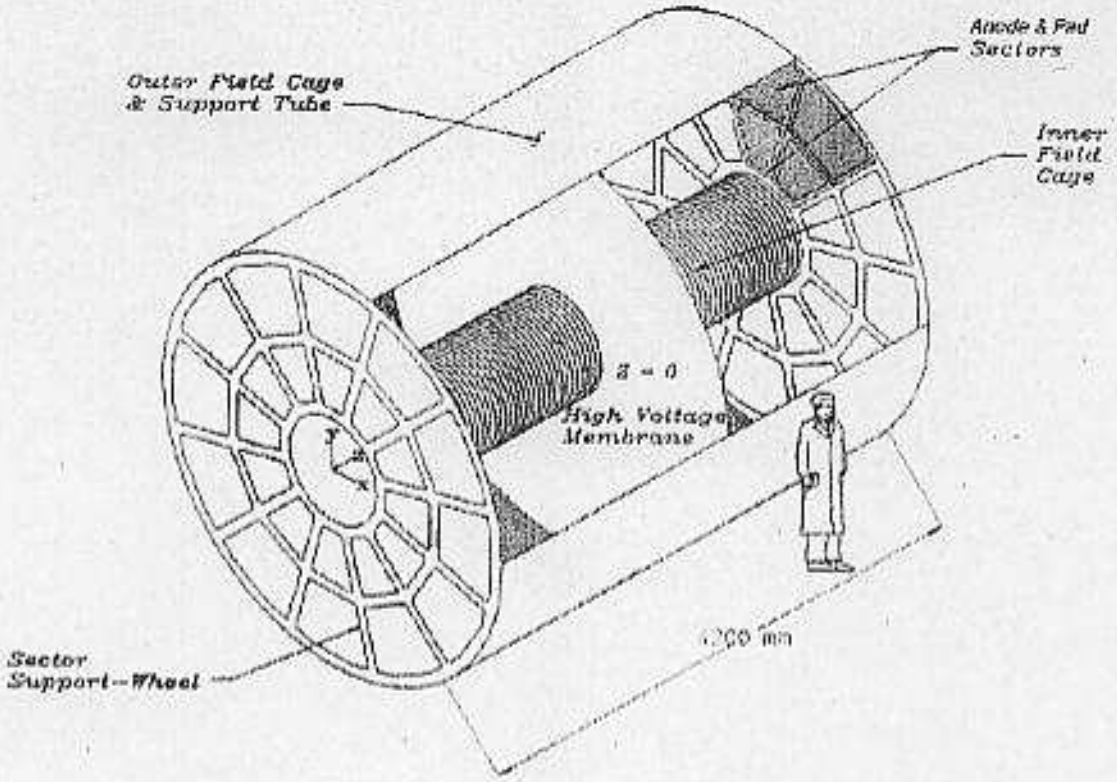

Fig. 2.3: Schematic diagram of the TPC.

2.3.1 The TPC Gas System

As mentioned, the TPC detector will be filled with P-10 gas as an ionization medium for the charged particles. The drift speed of an ionized electron depends not only on the electric field, but also on the type of medium it's going through. Thus, choosing the right gases and gas mixture as well as monitoring the gas content become very important.

A schematic arrangement of the gas system is shown in Fig. 2.3.1 [32] below.

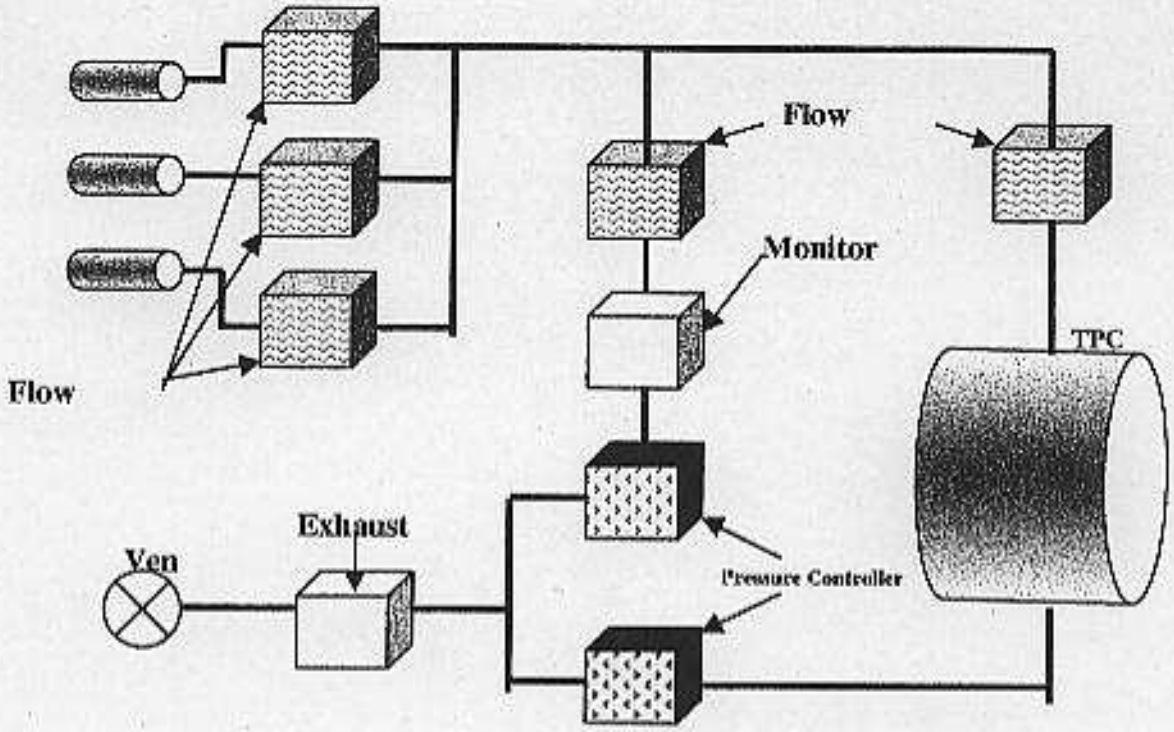

Fig. 2.3.1 A schematic diagram of the TPC gas system [35].

An Allen-Bradley Interlock System is installed in the gas room to monitor and control all the TPC subsystems. There are 16 green lights and 16 red lights on the panel representing the system status. Green stands for ON and red stands for OFF. By checking the color of the lights, the system status can be identified. The channels monitored are: Pioneer 1 through 4 (each represents the status of a methane sensor), Gas System & Computer Status, CH_4 Monitor Insulator, O_2 Monitor Insulator, Gas System Power, Laser System Enable, Gating Grid Enable, Anode High Voltage Enable, OK Signal to STAR Controls, Monitor Chamber Power and Cathode High Voltage Enable. The sixteenth channel is not used as this paper is written. Fig. 3.2.2.1 shows the Gas-Interlock Panel designed by the author.

There are four sensors attached to a small Pioneer computer. The sensor readouts are analog 4-20-mA current-loop readouts. The Pioneer computer reads the sensors, calibrates the signals relative to 100% of LEL (lower explosive limit), and applies a programmable threshold. In other words, the Pioneer issues an alarm if the methane gas in the gas system rises to a dangerous level (i.e. 10% of LEL). When it signals the alarm, it also sends a status signal to the Allen Bradley Interlock system. This status signal is a single on/off bit.

The gas-control program runs on a PC using the Windows NT operating system. Data is saved every 20 seconds on a cross-mounted disk which is accessed by Slow Controls. The values are displayed in EPICS. This is the only subsystem with a front-end not under the direct control of an EPICS interface. The implementation of an EPICS front-end was not cost-effective since the gas system was developed in Russia using interfaces developed specially for the STAR experiment.

2.3.2 TPC Field Cage

The TPC contains an outer and an inner field cage. They will provide a uniform electric field inside the TPC. The structures of the inner field cage and the outer field cage are nearly identical. A field cage consists of 182 rings and 183 resistors. Each ring is separated from the next by a 2 megohm resistor.

2.3.2.1 Choosing the Drift Field

The STAR TPC uses an externally applied electric field to drift the electrons from any point in the TPC to the anode and pad planes. The rate of drift is proportional to the applied field, but it isn't linear and it depends on the gas composition.

Fig. 2.3.1.2 is from Sauli's 1977 CERN lectures. Drift velocity is shown plotted versus the reduced electric field (i.e. electric field / gas pressure). STAR uses P-10 gas.

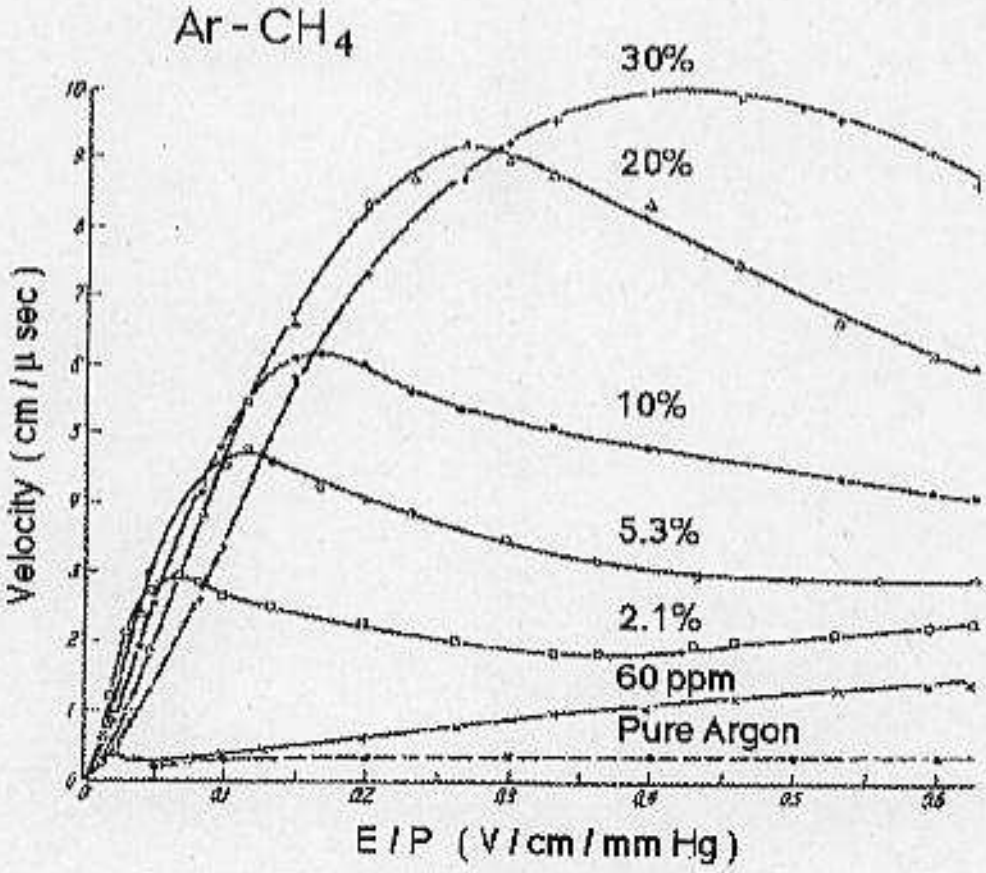

Fig. 2.3.1.2 Drift velocity vs. the reduced electric field

For the purpose of accurate track reconstruction, it is reasonable to choose an electric field near the peak in the drift velocity curve. This ensures that the drift does not change much and the drift velocity is least sensitive to minor changes in the gas pressure or temperature caused by the local environment. However, the STAR TPC has an automatic drift-velocity stabilization feedback loop which works by monitoring the drift of laser tracks in the TPC. Since the origin of these tracks is well-known in time and space, it is easy to calculate the actual drift velocity and to apply corrections to the external field to compensate for any time-dependent variations in the gas properties. The operating point for the drift velocity must therefore be slightly off the peak in order to provide some slope to the observed changes in parameters and to avoid the problem of a double-valued solution when the drift velocity is observed to drop[33].

Inspection of Fig. 2.3.1.2 reveals that any reduced field greater than 0.16 V/cm/mm-Hg satisfies these conditions. Assuming that operation will be near standard temperature and pressure, the drift field should be slightly greater than 120 V/cm. This is why the TPC is operated at an average voltage gradient of about 145 V/cm.

2.3.2.2 Tuning the field cage

The TPC field cage defines a uniform electric field between the central membrane and the gating grid near the pad plane. A high negative voltage will be applied to the central membrane and the two ends will be grounded.

Fig. 2.3.2.3 is the schematic diagram of the resistor chain in the field cage that creates a uniform electric field.

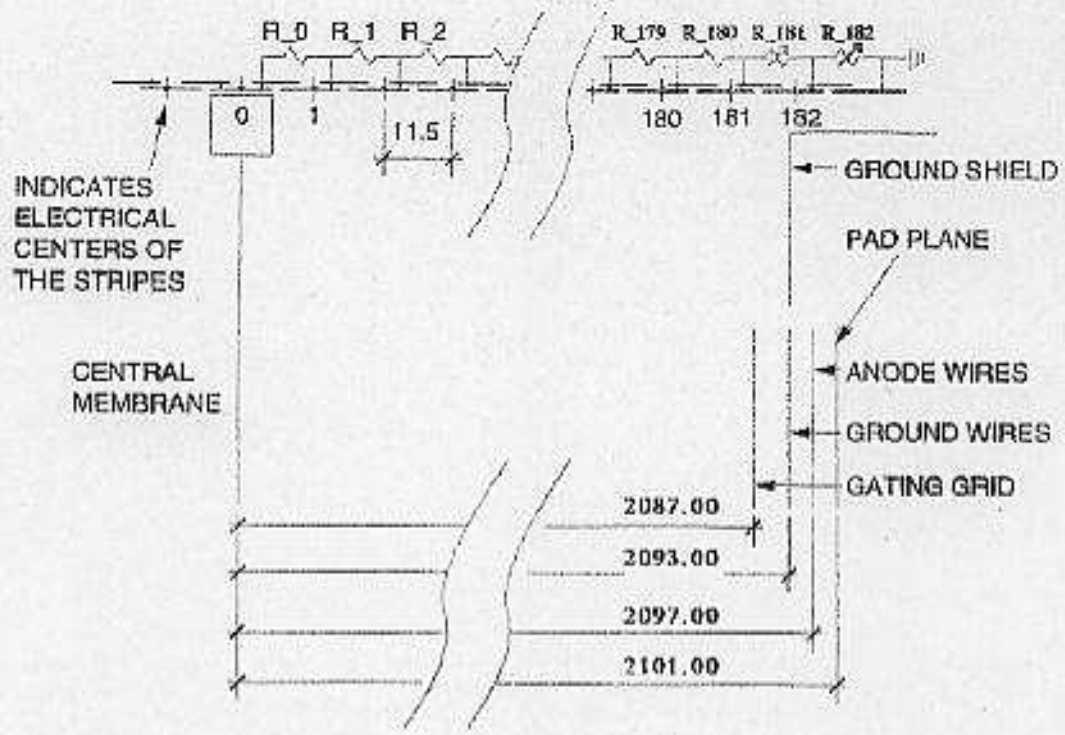

Fig.2.3.2.3: the schematic layout of the resistor chain (units in mm).

The terminus of the region of uniform drift speed is the gating grid. However, the TPC field cage extends beyond the gating grid in the region outside the anode sector and the parallel field is required to continue into this region to prevent distortions of the tracks at the sector boundary. Two adjustable resistors set the voltage on the last two field cage rings and they must be tuned to achieve this goal.

The resistor chains are different for the outer field cage and the inner field cage. The outer field cage has a "ground shield" attached to the last ring (182) and the shield is lined up with the "ground wires" in the anode sector. This prevents the shield from being located at the center of the ring and requires the ring to be set at the correct voltage for the ground shield, rather than its own natural setting. Accomplishing this requires tuning both resistors.

In order to easily adjust these two resistors, they have been removed from the

TPC and installed in an external rack. The resistor chain terminates in a scanning current meter so that the currents flowing in each chain can be monitored. This is a useful diagnostic for finding short circuits in the voltage-divider system. A scanning voltmeter also monitors the voltage on the ends of the two rings to ensure that they are set properly.

In the special case of the inner field cage where there is no ground shield, the settings on the final two resistors are fixed: R_{181} = 2 M Ohms, R_{182} = 310 K Ohms. By comparison in the outer field cage, we have: R_{181} = 1.870 M Ohms, R_{182} = 440 K Ohms. Detailed calculations of these values can be found in reference [33]

Chapter 3

STAR Slow Controls and the Author's Contribution

3.1 STAR Slow Controls

The STAR Slow Controls system sets, monitors and controls all the subsystems. Slow Controls ensures the validity and consistency of the recorded data so that physics may be extracted from it. A real-time, running VxWorks environment is utilized for this purpose and a software toolkit, Experimental Physics and Industrial Control System (EPICS), is used to operate all the controls.

3.1.1 Slow Controls Overview

An example of Slow Controls user interface for the TPC is shown in Fig. 3.1.1.

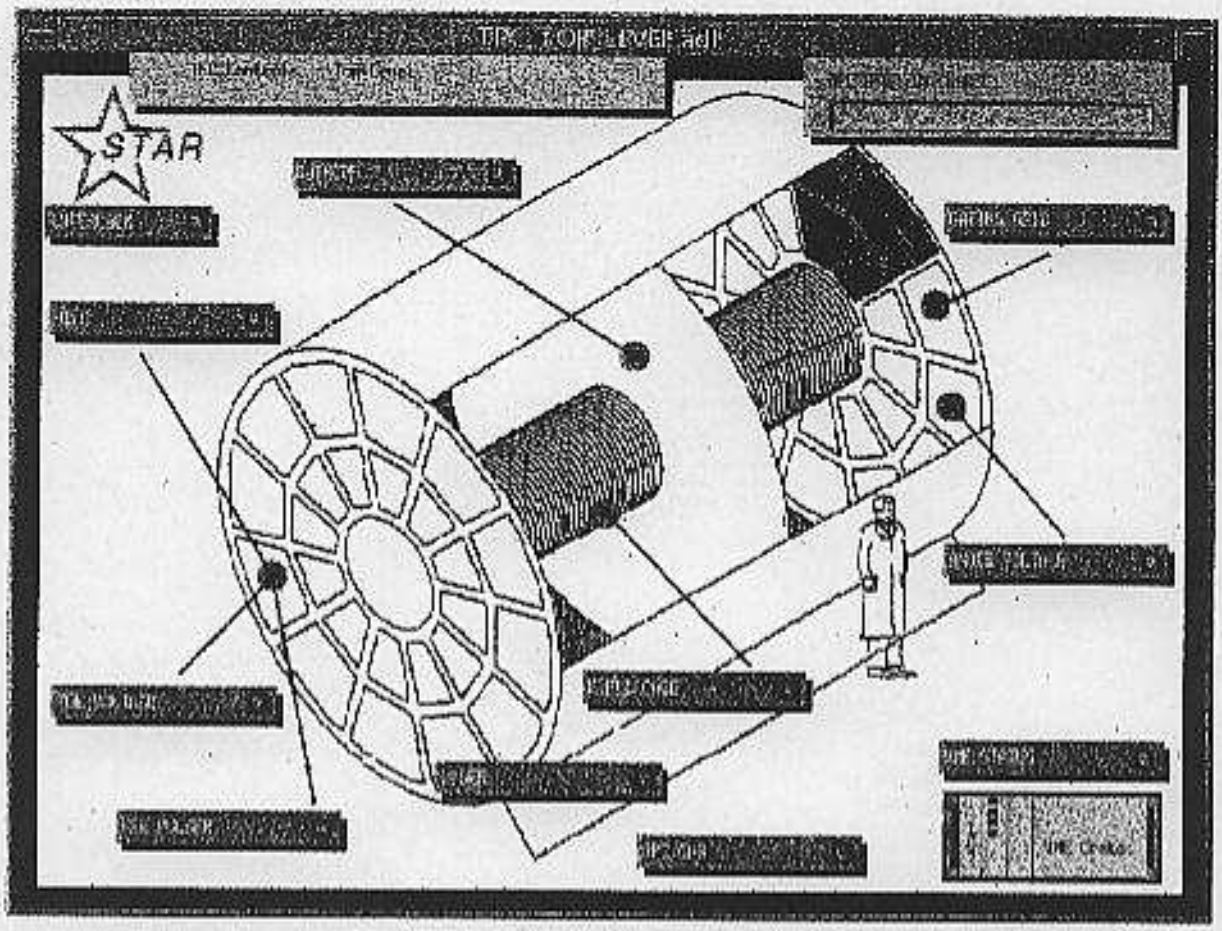

Fig. 3.1.1: The general control of the TPC

3.1.2 Experimental Physics & Industrial Control System (EPICS)

EPICS was selected as the foundation for the STAR control software environment because it incorporates a standard means of sharing information and services, graphical display and control interfaces, and a robust, real-time operating system. EPICS was designed by Los Alamos National Laboratory and the Advanced Photon Source at Argonne National Laboratory as a development toolkit.

The EPICS run-time environment consists of several integrated components that share a standard communication layer. Together, these components provide a system for

the data collection, supervisory control, continuous control, and sequential control, along with interfaces to modeling and analysis codes. The components of EPICS used by STAR are:

- Motif Editor and Display Manager (MEDM), an operator interface that provides full-color, window-based screens that can mimic control panels.

- Graphics Database Configuration Tool (GDCT), the design tool for a distributed, run-time database that isolates the hardware characteristics of the I/O devices from the applications program and provides built-in, low-level control options.

- A sequencer that implements state-based control in order to support system automation.

- An alarm manager to organize system-wide alarms for quickly pointing to problems without overloading the operator.

- A data archiver that acquires and stores run-time data and retrieves the data in a graphical format for later analysis.

- An interface to a channel access program establishes a network-wide standard for accessing the run-time database.

All functions in the system are optimized for speed and efficiency. For example, event-driven mechanisms are used throughout the system to reduce or eliminate the need to poll the data sources. Fault detection mechanisms are built-in at all levels to provide a robust system. Fault detection is used by all the EPICS run-time components and is available to the application programs. System-wide data synchronization correlates data with a particular system event, such as a beam pulse [34].

In order to run EPICS, a VME processor is required. It loads the designated database files and establishes all the hardware links designed in the database. It then begins to send the control commands to the hardware that is linked to the VME processor. Meanwhile, the processor will also be monitoring the commands throughout the network and respond to data requests. The response will then be returned and displayed on the remote user's screen by the MEDM software.

A brief introduction to EPICS follows; more detailed examples will be given when the author's contribution is described.

3.1.2.1 Motif Editor Display Manager (MEDM)

The MEDM is a Graphical User Interface (GUI) for controlling the hardware and retrieving data. Each object in MEDM is associated with a record name from GDCT. By choosing a certain object, MEDM will broadcast the command or request to the whole network. Once the request is acted on by the appropriate VME processor, the information is sent back to the MEDM screen for display. This may generate an alarm, depending on the database design.

3.1.2.2 Graphical Database Configuration Tool (GDCT)

In this database design tool, there are many records written in the C language that perform all the desired functions, such as input/output. It utilizes a Graphics User Interface (GUI) to ease the burden on the programmer. Each record must have a unique name which acts like a variable name in the C language. By specifying record fields, the programmer can tell the record to which hardware device it will be linked and the frequency of its execution. Each record performs a certain task. An EPICS programmer must choose the necessary records and create necessary limits; this takes some experience.

GDCT creates and maintains two files: a graphical information file and a ".db" file. The ".db" file is an ascii file that contains only the information that EPICS requires to create a database. This information includes only EPICS records and fields. The graphical information file contains the graphic representation of records and links to and between the records. There are some default database settings in the default.dctsdr file which must be loaded before loading any user-designed databases.

3.1.2.3 State Notation Language (SNL)

The State Notation Language is used to perform some functions that cannot be included or cannot be easily handled in the GDCT design. It is a text-format program and must be compiled and loaded into the VME processor. The variable names inside a SNL program

are linked with corresponding record names, so that whatever actions taken on that variable will take effect in the corresponding record. The priority of SNL is higher than that of GDCT, so it can read and change all the default value settings in GDCT.

3.1.2.4 Alarm Handler

The Alarm Handler contains all the records that need and have an alarm status. The safe operating limits of each variable which is associated with a record name can be set inside the database. When the monitored variable exceeds the limits for that variable, EPICS will generate an alarm for that record and send the alarm to the Alarm Handler. There are two stages of alarm, major and minor. If it is a major, severe alarm, the name bar will blink in red; if it is a minor alarm, the name bar will blink in yellow. If provided with a speaker, the user will hear a beep as the name bar blinks. The name bar appears gray when there is no alarm. In this way, the Alarm Handler identifies the source of trouble and subsequent alarms and makes proper notification so that prompt corrective action may be taken.

3.1.2.5 Summary

EPICS is unique in the way that it can communicate with the hardware in real time. Once the database written for a certain purpose is loaded, a command can be sent through the network. This design eliminates the need for the user to know the intimate details of how EPICS works. All the user must do is send commands and requests.

3.2 Author's contribution

Since joining the STAR Slow Controls group, the author has been actively involved in the EPICS design of several different projects including: MEDM screen modification, an alarm handler and archive tool for the Field Cage Current (FCC); all the EPICS programs for the TPC Gas-Interlock system; all the EPICS programs for the LeCroy High-Voltage Power Supplies for the TPC Trigger group, and the EPICS monitor for the TPC Plane Pulser.

3.2.1 Field Cage Current (FCC)

As mentioned in section 2.3.2.2, the TPC field cage extends to the region outside the anode high-voltage sector, so the parallel field must continue into this region. The last two resistors set the voltage on the last two field cage rings and they must be tuned for this purpose.

A user interface was designed to monitor the current and voltage on those two resistors. It generates an alarm whenever the current or voltage exceeds set limits. Slow Controls is responsible for the monitoring but not the control of field cage current and voltage.

The original program for the FCC was written by Dr. Jan Chrin. The author was involved in some modifications of the MEDM screen and the addition of an alarm handler, which will be shown in the following sections. It is currently up and running.

3.2.1.1 MEDM screen for FCC

Fig. 3.2.1.1 Shows the MEDM user screen used for FCC monitoring.

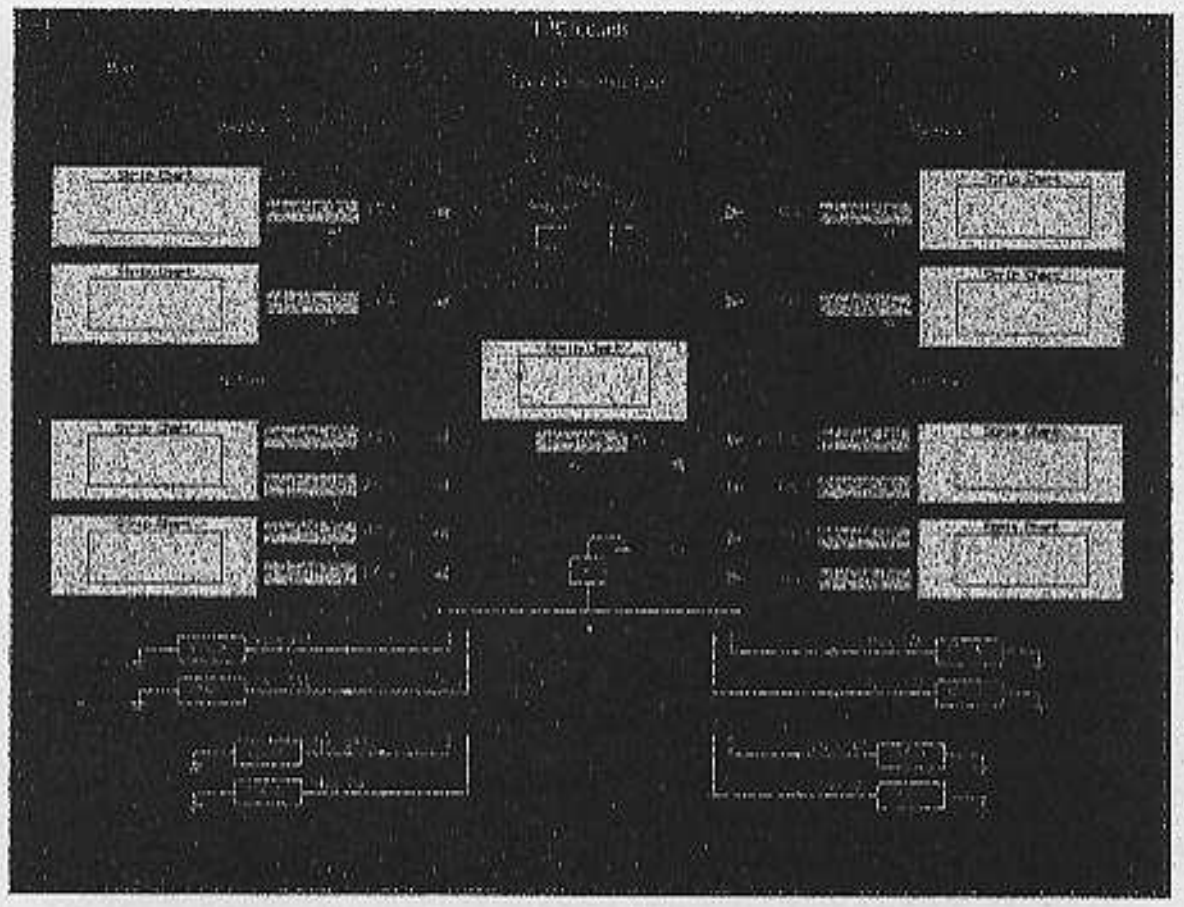

Fig. 3.2.1.1: The MEDM screen for FCC

The two buttons in the upper middle are the control buttons for displaying a GPIB device. The displays in the middle show the current and voltage data at the two ends of the inner/outer field cages. The schematic diagram on the bottom of the screen shows the location of the two resistors that are being monitored.

3.2.1.2 Alarm Handler for FCC

Fig. 3.2.1.2 shows the Alarm Handler used in the Field Cage Current (FCC) program. When the current or voltage exceed set limits, an alarm will be generated. It appears as a blinking name bar and/or a beep. For example, if the voltage of the "0 ring" on the west side of the outer field cage exceeds the major limits given in the database configuration, the name bar with "TpcFCCVW_OFC_0" will blink in red and an audible beep will be generated.

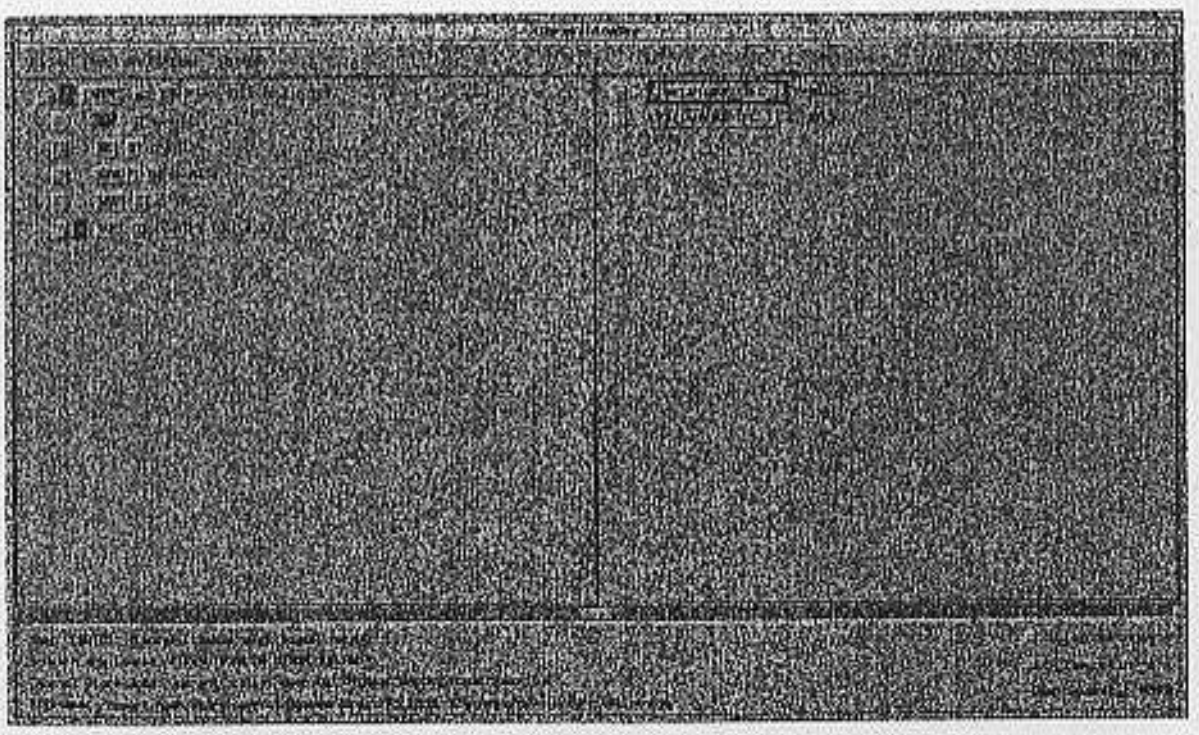

Fig. 3.2.1.2: The alarm handler for FCC

3.2.2 Gas-Interlock System

As part of the TPC system, the gas system was commissioned for a test run in November, 1998. A gas-interlock system is installed to control all the subsystems: Laser, Gating Grid, Anode High Voltage, STAR Controls, and Cathode High Voltage.

All the subsystems are linked together. For example: in order for the Gating Grid subsystem to operate, the gas system and the computer must be ready, so it is very important for the operator of each subsystem to know the status of the main gas system. The gas-interlock panel is installed in the gas room, so it cannot be easily accessed during full TPC operation. In order for people at a remote location (the experiment control room) to be notified of the status of all the subsystems, it is imperative that the gas-interlock system information be broadcast online in real time. This motivated the construction of the Slow Controls Gas-Interlock Panel.

3.2.2.1 MEDM screen for Gas-Interlock System

Fig. 3.2.2.1 shows the MEDM screen for the Gas-Interlock Panel. In this case, only the status of fifteen channels (green lights) are sent to Slow Controls due to a hardware limitation. The logic diagram in this figure shows a "green-light logic." Only when the status of the methane sensor (Pioneer 1 through 4) are all ok, the gas system power can be turned on. Only after the gas system is turned on and the computer that is monitoring the gas system is ready, can all the subsystems (laser, gating grid, anode high voltage, STAR controls, chamber power) be turned on (not necessary if you do not enable those subsystems). Only when the status of methane and oxygen are both ok can the cathode

high voltage be turned on. The lights in the far-right column are reserved for future use.

There is also an "override" feature of the Gas-Interlock Panel. With a key, the operator can manually turn on the green light without the preliminary green lights on. This results in a blinking green light. Red-light status is the logical opposite of the green light. Whenever the green light is on, the corresponding red light will be off, and vice versa, except when the green light is blinking. This "green logic" reflects the real logical relationship between subsystems indicated by a light.

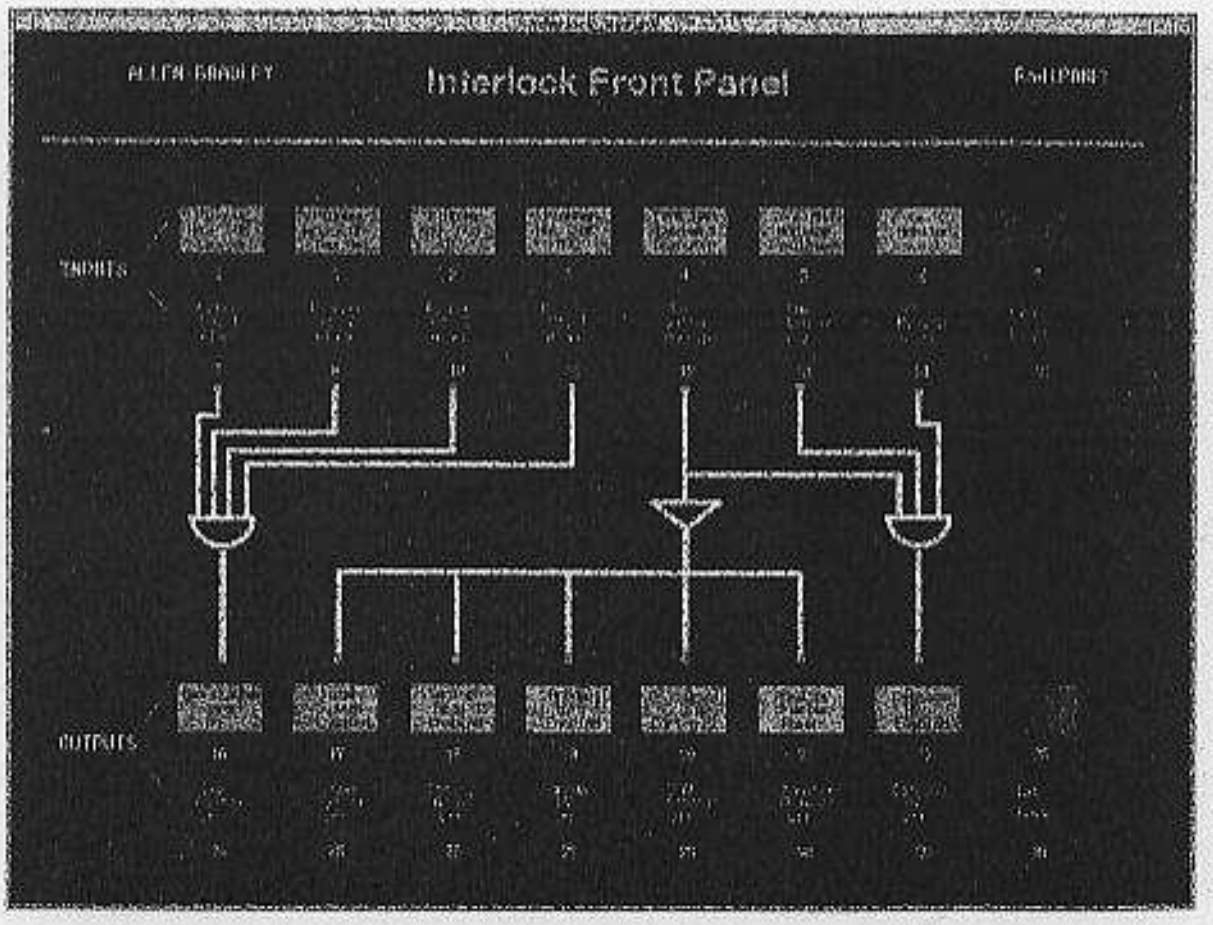

Fig. 3.2.2.1: MEDM screen for the interlock panel by author.

3.2.2.2 GDCT database for Gas-Interlock System

The status of the sixteen channels (the logic inputs) is sent to Slow Controls via TTL data lines. Five volts represents the "on" status, and 0 volts represents the "off" status of the green light. This voltage information is received by the Acromag AVME 948X 64-bit I/O module and converted into "1" or "0" binary format by a TTL circuit which is then output to the I/O module. Thus, the GDCT records will be able to read the binary values through the I/O module. In this project, the light status is updated every second. The module pin-out is presented in Appendix 2.

There is a hidden problem in the signal display. As previously mentioned, the green light could be blinking by the "force-on" feature. In this case, the corresponding red light should be off. Since we do not have any information on the red light, if we do not make any modifications, the red light will be blinking as well, instead of staying red. To get around this, more records are added to the database and further judgement about the light status is made by the State Notation Language program which will be discussed in the next section.

Fig. 3.2.2.2 shows the database configuration for the Gas-Interlock System.

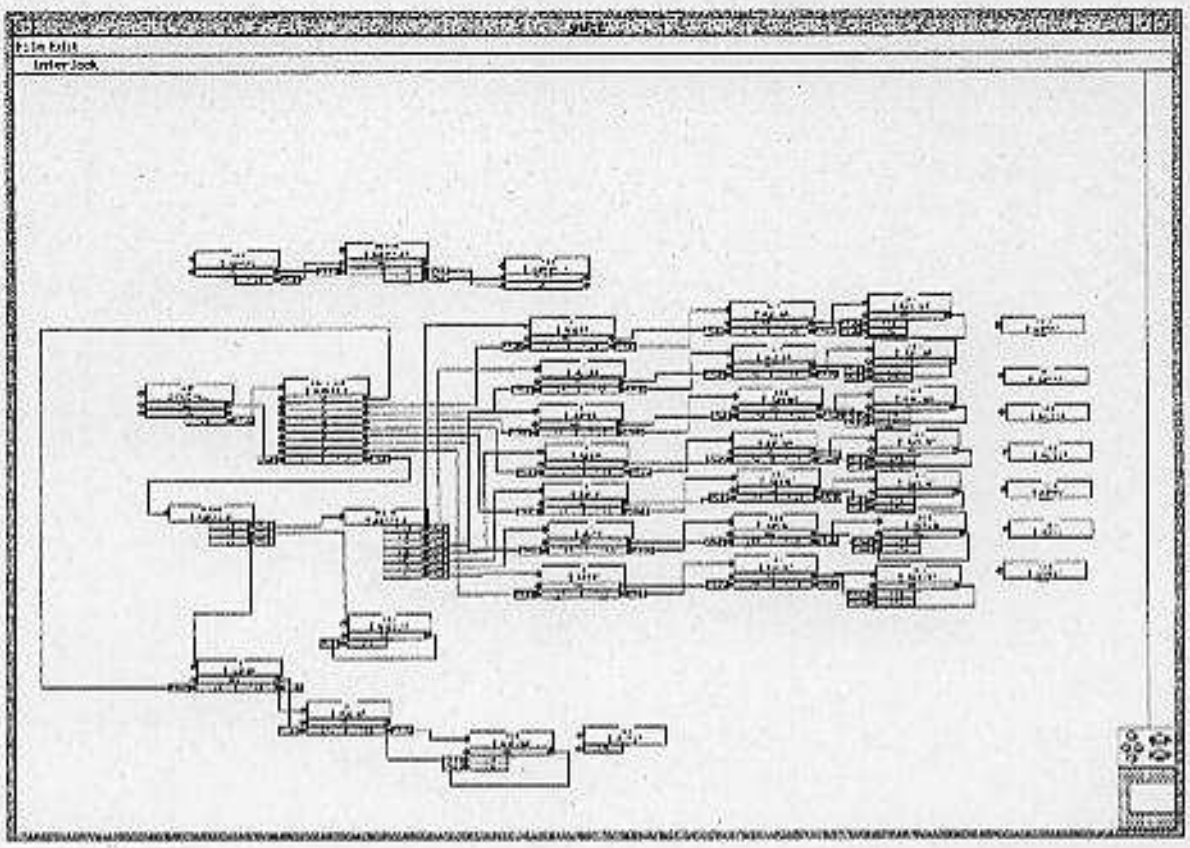

Fig. 3.2.2.2: Database for the Gas-Interlock System

3.2.2.3 Sequence File for Gas-Interlock System

As was mentioned in the previous section, there is a problem of eliminating the blinking red lights on our screen when the channel is being overridden. To solve this problem, not only must the information about the channel logic state be obtained for the channel, it must also be known whether the channel remains in that state. So one more record is added for each channel to distinguish the "blinking" status from the normal status.

All these decisions are accomplished in a sequencer program. A copy of the interlock sequencer program is shown in Appendix 3.

3.2.2.4 Alarm Handler for Gas-Interlock System

When the whole TPC is operating, the sixteen status bits should all be set. A bit which is not set signals that something is wrong with the device that it represents. An alarm should be generated to inform the operator and all other gas-system users. In fact, a loud alarm provides this warning at the experiment, but it would not be able to inform people at a remote location. Another alarm handler is needed for this purpose.

Fig. 3.2.2.4 shows the alarm handler built for the Gas-Interlock System.

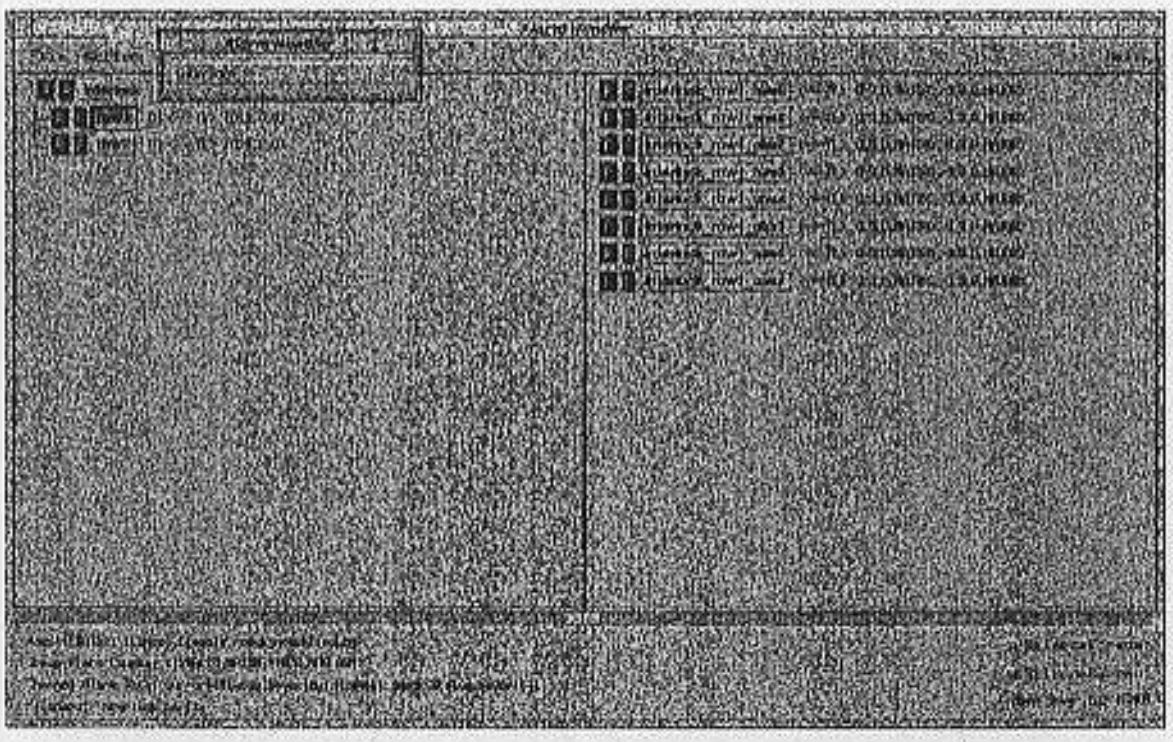

Fig. 3.2.2.3: Alarm Handler for the Gas-Interlock system.

3.2.3 LeCroy HV (High Voltage) for Trigger Group

Based on rapidly digitized distribution information for each RHIC beam crossing, the trigger system determines whether or not to initiate the recording of a particular event. A LeCroy 1440 high-voltage power supply is used to provide high voltage to the phototubes inside the Central Trigger Barrel (CTB) detector. Hardware Controls monitors and sets the voltage on each channel of the LeCroy 1440 via an RS232 serial connection. Communication is accomplished using subroutine records. New subroutine functions were written for this purpose.

3.2.3.1 MEDM screen for LeCroy 1440A

The MEDM screen for the LeCroy 1440A High-Voltage Power Supply is shown in Fig. 3.2.3.1.

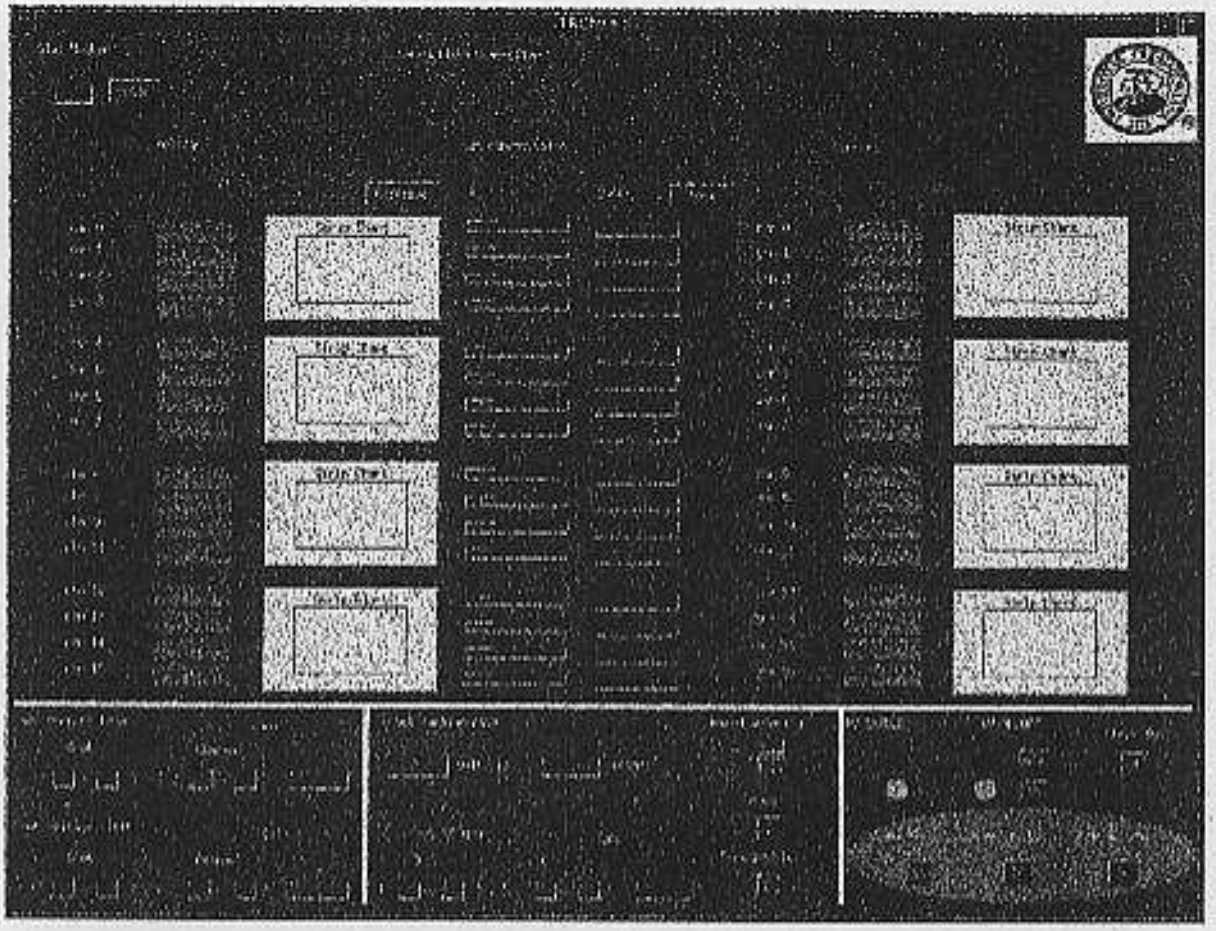

Fig. 3.2.3.1: MEDM screen for LeCroy 1440A

The LeCroy 1440A controller has a programmable chip inside which can be remotely controlled. This controller sets, monitors and controls all sixteen Lecroy 1446 high-voltage boards in the LeCroy High-Voltage Power Supply crate. In the current Slow Controls version, the MEDM screen controls the LeCroy 1440A controler to determine the high voltage being supplied to the trigger detector.

All the initial settings that need to be placed for LeCroy 1440A are on the control

panel at the bottom left of the screen. In the bottom right of the screen are the control buttons, such as the power switch and the feature display.

A minor problem with this screen is that much information must be displayed in other UNIX windows due to the limitations of MEDM. The data coming from the serial port is in string-with-line-break format, which is not currently supported for a MEDM screen.

3.2.3.2 GDCT Database for LeCroy 1440A

Due to the limitations of EPICS, most of the work in creating the link between EPICS and the LeCroy must be done in an external C program. EPICS acts as a bridge between the serial-port command and the final data output on the MEDM screen. An EPICS subroutine was used for this purpose. The subroutine record has a C language-like structure. Field "d" of the structure is used to control the loop for each function. Fields "e" to "l" are used to store the data retrieved from the serial port, and the remaining fields are used to store the commands. The operations are then done by the C program, which will be discussed in the next section.

Fig. 3.2.3.2 shows the database configuration for the LeCroy 1440A.

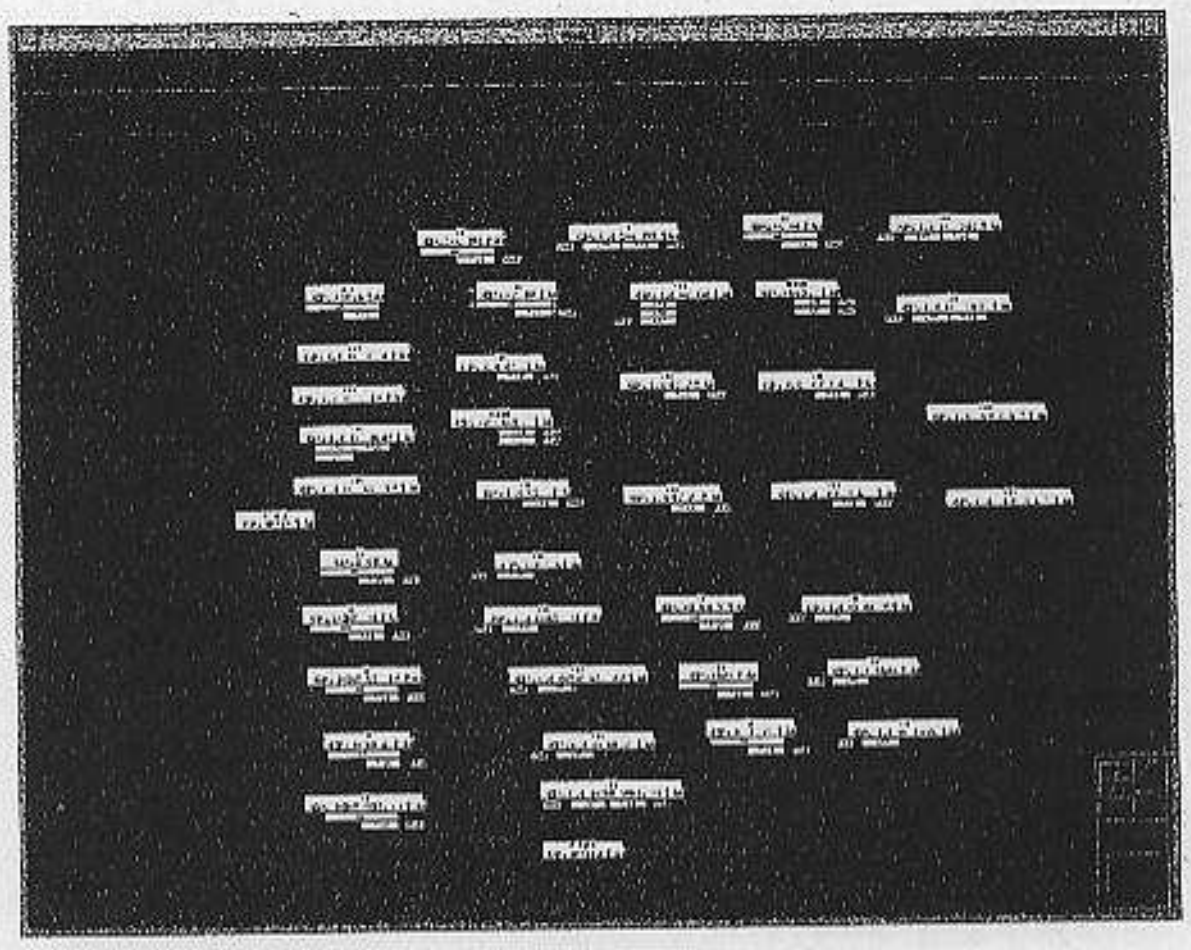

Fig. 3.2.3.2 Database configuration for the LeCroy 1440A

3.2.3.3 C program for LeCroy 1440A

The loop through each function of this C program is controlled by a message queue stored in field 'd' of each subroutine record. That record will be processed only if the value of the field is 1. In each switch statement, certain functions are performed, such as sending commands and getting data through the serial port.

Another important task is the establishment of a serial connection between the VME processor and the LeCroy 1440A. This link is established by opening a file stream in the VxWorks environment [35]. Once this file is successfully opened, the serial port is opened. All commands can then be written to this file and actions can be performed on the hardware, either to control the device or request data from it.

The object file of this C program is loaded during the VME processor start-up. A copy of this program is included in the Appendix 4.

3.2.4 Wavetek Monitor for Plane Pulser

The Wavetek 350 is a waveform generator. It generates simulated data signals during the configuration stage for the Gating Grid system to test that the pad generates the correct corresponding signal. At the start of a run, the Wavetek is disconnected.

During accelerator operation, there is no physical access to the Wavetek, so Slow Controls manages the control of the Wavetek through an RS232 serial connection.

3.2.4.1 MEDM screen for the Wavetek 350

A sample wave that is loaded into the Wavetek is shown in Fig. 3.2.4.1a.

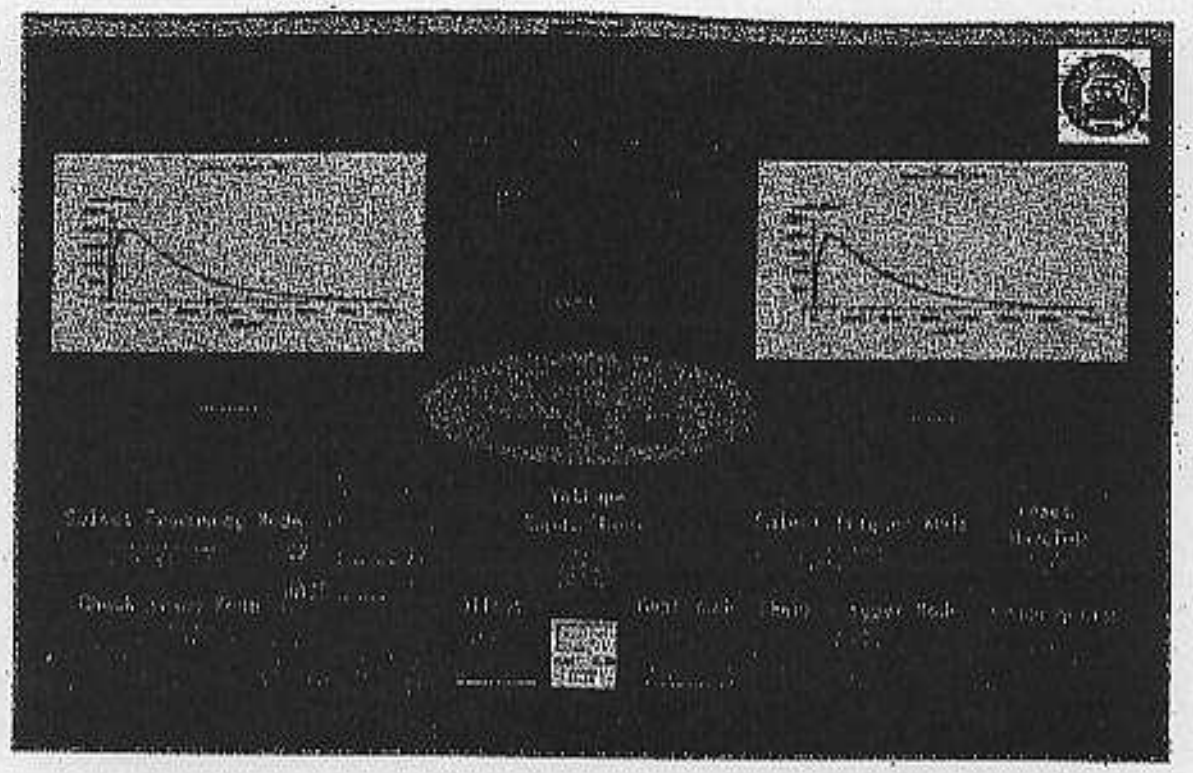

Fig: 3.2.4.1a Monitor MEDM screen for the Wavetek 350

On the left side is the input wave and on the right side is the output data from the Wavetek. These two pictures are displayed together to allow comparison between the input and output signals.

There is also a related screen shown in Fig. 2.3.4.1b that performs the task of making I/O connections with the Wavetek, such as "define a new trace," "delete a trace," etc.

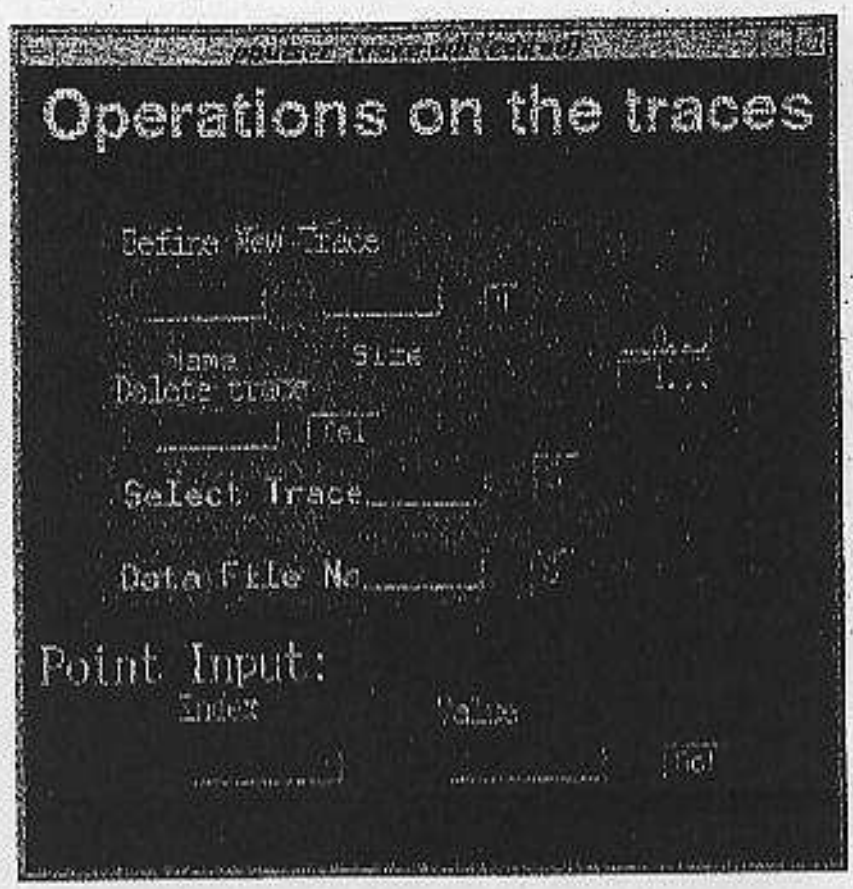

Fig. 3.2.4.1b MEDM screen for the trace operation on Wavetek

3.2.4.2 GDCT screen for the Wavetek 350

The database design for the Wavetek is relatively easy compared to the LeCroy program. There are virtually no connections between the records because each record represents an individual function dealing with a separate task.

Fig. 3.2.4.2 below shows the database design.

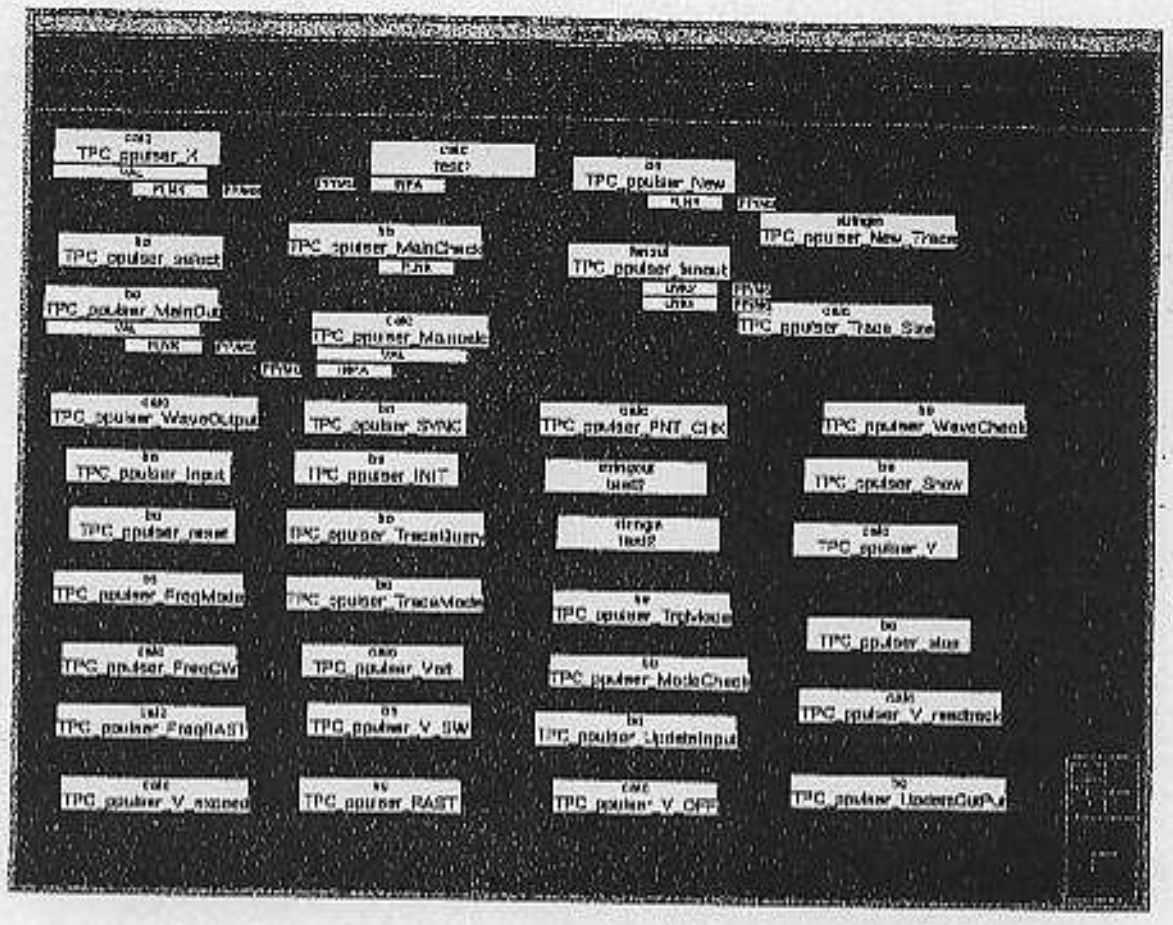

Fig. 3.2.4.2 Database for Wavetek350

3.2.4.3 SNL code for the Wavetek 350

In principle, the Wavetek program is the same as the LeCroy control program. But instead of using a subroutine record in GDCT and depending on the C code, the State Notation Language (SNL) is utilized; a recommended procedure in EPICS. The source code appears in Appendix 5.

3.2.5 Future Plans

As the time approaches for the STAR commissioning run, the programming work is reaching its final stage as well. The next task will be to maintain the programs. This is the biggest problem in computer programming. Not only does the program have to be kept running, it has to be understood by the follow-up people in case there are changes to be made.

A second major task facing STAR Slow Controls is to upgrade EPICS to a new version. Already the incompatibility of the two versions of the sequencer poses a problem in one of the Slow Controls workstations. Some programs are not platform or device independent. Eventually EPICS technical support will stop supporting the EPICS SUN-OS version and focus on the Solaris version. All code will need to be ported to this platform. These are critical problems which must be addressed.

Conclusion

An EPICS-based control system has been developed for the STAR experiment at RHIC. The system developed contains a number of new EPICS implementations. Controlling and monitoring the STAR hardware through a network is not a easy job, especially when real run-time control is required. EPICS, running in a VxWorks environment, seems to be appropriate for this purpose. It is reliable and easy for the operator to use. It does not require the programmer to be very well-acquainted with C language programming, but when EPICS is unable to accomplish certain jobs, C experience is very helpful. Using it, Slow Controls has been able to fulfill its mission of monitoring and control during test runs of the TPC; Slow Controls is poised for STAR's up-coming commissioning run.

Appendix 1 Rapidity and Pseudorapidity

Rapidity

Rapidity is a dimensionless measurement of a particle's forward momentum along the beam line:

$$\gamma = \frac{1}{2}\ln\left[\frac{E + p_z c}{E - p_z c}\right]$$

where E is the particle's total energy and p_z is the momentum along the z-axis (along the beam line).

Pseudorapidity

Pseudorapidity (η) is used in high energy physics to measure a particle's trajectory relative to the beam line. This is the rapidity assuming a massless incoming particle.

For particles traveling at nearly the speed of light, we have

$E = p\,c$,

$p_z = p\cos\theta$

Substitution into the rapidity definition gives the following

$$\eta = \frac{1}{2}\ln\left[\frac{1+\cos\theta}{1-\cos\theta}\right]$$

which can be reduced to:

$$\eta = -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]$$

θ is the angle between the particle's momentum p and the beam line. The pseudorapidity for various directions in the STAR detector are shown in figure A1.

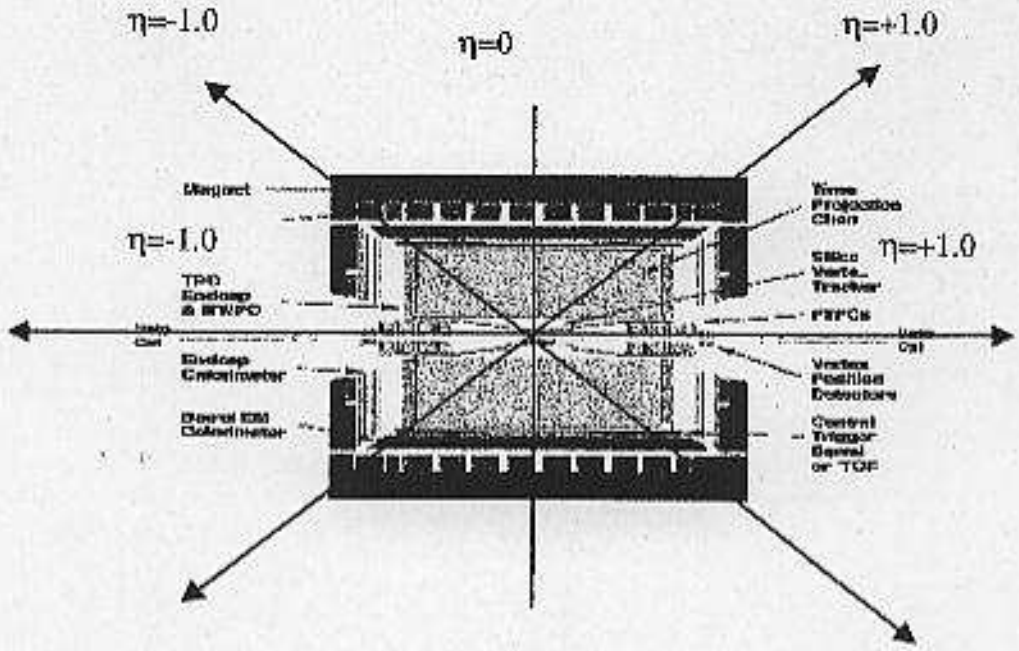

Fig. A1

Appendix 2 Pin-out for the Gas-Interlock System

There are altogether 64 pins on the Acromag AVME module, only 32 of which can be used to read data. So sixteen pins must be connected to the sixteen outputs for the green-light status, i.e. subsystem status.

Table A2 shows the connection of the output connector. The P3 pin number indicates the number of pins on the AVME 948X 64-bit I/O module. The Base address is set by the jumpers on the module. It is currently set to 0X3000.

Memory Address (Base Address +)	Bit Number	P3 Pin number
256	7	33
	6	35
	5	37
	4	39
	3	41
	2	43
	1	45
	0	47
257	7	34
	6	36
	5	38
	4	40
	3	42
	2	44
	1	46
	0	48

Table A2. The Pin-Out of the Acromag I/O module for interlock.

Appendix 3 SNL code for Interlock System

```
program interlock_scan

short count;
assign count to "interlock_row2_count";
short idle;

short sum0;
assign sum0 to "interlock_row2_sum0";
monitor sum0;
short sum1;
assign sum1 to "interlock_row2_sum1";
monitor sum1;
short sum2;
assign sum2 to "interlock_row2_sum2";
monitor sum2;
short sum3;
assign sum3 to "interlock_row2_sum3";
monitor sum3;
short sum4;
assign sum4 to "interlock_row2_sum4";
monitor sum4;
short sum5;
assign sum5 to "interlock_row2_sum5";
monitor sum5;
short sum6;
assign sum6 to "interlock_row2_sum6";
monitor sum6;

short sum7;
assign sum7 to "interlock_row2_sum7";
monitor sum7;

short light0;
assign light0 to "interlock_row2_red0";
short light1;
assign light1 to "interlock_row2_red1";
short light2;
assign light2 to "interlock_row2_red2";
short light3;
assign light3 to "interlock_row2_red3";
short light4;
assign light4 to "interlock_row2_red4";
short light5;
assign light5 to "interlock_row2_red5";
short light6;
assign light6 to "interlock_row2_red6";
short light7;
assign light7 to "interlock_row2_red7";

ss interlock_scan{
```

```
state init
{
        when(delay(0.2)) {
                count=0;
                pvPut(count);
                pvMonitor(count);
        }state no1

}

state no1
{ when(count !=15){idle=1;
        }state no2
}

state no2
{
        when(count == 15) {
                        if (sum0!=0){
                                light0=1;
                                pvPut(light0);
                                sum0=0;
                                pvPut(sum0);
                        }
                        else {
                                light0=0;
                                pvPut(light0);
                        }
                        if (sum1!=0){
                                light1=1;
                                pvPut(light1);
                                sum1=0;
                                pvPut(sum1);
                        }
                        else {
                                light1=0;
                                pvPut(light1);
                        }
                        if (sum2!=0){
                                light2=1;
                                pvPut(light2);
                                sum2=0;
                                pvPut(sum2);
                        }
                        else {
                                light2=0;
                                pvPut(light2);
                        }
                        if (sum3!=0){
                                light3=1;
                                pvPut(light3);
                                sum3=0;
                                pvPut(sum3);
                        }
```

```
                else {
                        light3=0;
                        pvPut(light3);
                }
                if (sum4!=0){
                        light4=1;
                        pvPut(light4);
                        sum4=0;
                        pvPut(sum4);
                }
                else {
                        light4=0;
                        pvPut(light4);
                }
                if (sum5!=0){
                        light5=1;
                        pvPut(light5);
                        sum5=0;
                        pvPut(sum5);
                }
                else {
                        light5=0;
                        pvPut(light5);
                }
                if (sum6!=0){
                        light6=1;
                        pvPut(light6);
                        sum6=0;
                        pvPut(sum6);
                }
                else {
                        light6=0;
                        pvPut(light6);
                }
                if (sum7!=0){
                        light7=1;
                        pvPut(light7);
                        sum7=0;
                        pvPut(sum7);
                }
                else {
                        light7=0;
                        pvPut(light7);
                        sum7=0;
                        pvPut(sum7);
                }

        }state no1
    }

}
```

Appendix 4 C code for LeCroy 1440A

```
/*jlin tpchv_arc.c */
/*
  Skeletal code for subroutine record served to pass lrs1458 commands.

  author: Heng Liu
  modified for using lrs1458 and arcnet    Wei-Ming Zhang KSU 07/02/96
  modified for lrs1461 with new UGI and rtdb  WMZ          12/31/97
  modified for interface with serial port    WMZ          8/26/98
*/

/*anhv_serial.c
  Skeletal code for subroutine record served to pass 1445A commands.
   author: Jie Lin (archer) 11/09/98 */

/* vxWorks includes */
#include      <vxWorks.h>
#include      <string.h>
#include      <stdlib.h>
#include      <stddef.h>
#include      <types.h>
#include      <dbDefs.h>
#include      <dbAccess.h>
#include      <dbCommon.h>
#include      <stdioLib.h>
#include      <msgQLib.h>
#include      <ioLib.h>
#include      <logLib.h>
#include      <stdio.h>
#include<time.h>
/* #include      <timers.h> */
#include      <tickLib.h>
#include      <kernelLib.h>
#include      <taskLib.h>

/* epics includes */
#include      <recSup.h>
#include      <devSup.h>
#include      <rec/subRecord.h>

/* ************** define for leps server  ********** */


#define CHK_SYS      0
#define POWER        1
#define READ_V       10
#define SET_DEMAND   22 /* this has been transferred to another proc.  */
#define READ_I       23
#define SET_RAMP     25
#define SET_VLIM     26
```

```
T_ITRIP     27
T_ILIM      28
T_READY     29
T_CLEAR     30
OW_LIMITS   31
OW_AC_TRIP  32
T_AC_TRIP   33
OW_VI           35
IP_STAT1    36
IP_STAT2    37
AD_V1       41
AD_I1       42
T_DEMAND1   43
IP1_STAT1   44
IP1_STAT2   45
OW_MODULES  46
LARITY      47

.X_NODES      1   /* MAX=2 0:outer & 1:inner   */
.X_SLOTS      1   /* MAX=12 (x8=96 channels) */
MAX_CHANS     8  */ /* MAX = 8  */
.X_CHANS     16  /* L1443 has 16 channels. */

AL_stat=0;
y_stat=0;
ILE_stat=0;
'_stat=0;
:R_stat=0;
s();
=7;  /* LeCroy1444 has 8 channels. */
=15; /* LeCroy1443 has 16 channels. */

Queue driven by Subroutine Records */


Queue driven by Subroutine Records */
 leps_Q;

outines: init and proc *** */

itialization. To execute it:  INIT_BIT ORD in server task
 each sub-record as we do in the rtdb for this program   */


(struct subRecord * psub)

IG
sInit was called by <%s>\r\n", psub->name);



 process sub-records. */
```

```
long lepsProc(struct subRecord * psub)
{
   if (psub->pact)
   {
/* Second pass */
    if (psub->udf)
      return (-1);  /* error return */
          return (0);                    /* processing complete */
   }
   else
   {
/* First pass */
    if (msgQSend(leps_Q, (char *) &psub, 4, NO_WAIT, MSG_PRI_NORMAL) == ERROR)
    {
     logMsg("Queue error in lepsProc()\n",0,0,0,0,0,0);
     psub->udf = TRUE;
     return (-1);  /* error return */
    }
          return (1);                    /* asynch processing */
   }
}
/*  Main routine to perform actions after corresponding sub-records
   defined in rdb are  processed                                  */

long leps_server()
{
   struct subRecord *psub;
   struct rset *pset;

   int    status;
   int    polarity=0;
   int    islot, i, j;

   void check_sys();

  /*  void settings(); */

   int Vlim_s_i=0, Vlim_s_f=0, Vlim_c_i=0, Vlim_c_f=0, Vlim=0;
   int Ramp_s_i=0, Ramp_s_f=0, Ramp_c_i=0, Ramp_c_f=0, ramp=0, ramp_time=1;
   int step=10;
   int Itrip_s_i=0, Itrip_s_f=0, Itrip_c_i=0, Itrip_c_f=0, Itrip=0;
   int AC_trip_s_i=0, AC_trip_s_f=0, AC_trip_c_i=0, AC_trip_c_f=0, AC_trip=0;
   int trip_stat1[MAX_CHANS]={0}, trip_stat2[MAX_CHANS]={0};
   char bufout_VI[800]="\0";
   char tempI[MAX_CHANS][50]={"\0"};
   char tempV[MAX_CHANS][50]={"\0"};
   float  demand[MAX_CHANS]={0}, demand_old[MAX_CHANS]={0} ;
  /* float voltage_now[MAX_CHANS]={0}; */



   leps_Q = msgQCreate(1000, 4, MSG_Q_FIFO);    /* create msg Queue */

   /* open a file stream */
```

```
  fd = open("/tyCo/1", O_RDWR, 0644);


 /*
  * initialize serial line and find out if anything is there
  */

  /*  rate= ioctl(fd, FIOBAUDRATE, 9600); */
   status = ioctl(fd, FIOSETOPTIONS, OPT_RAW);

   if (status == ERROR)
   if(status==0)
      return (-1);
   else printf("serial port opened OK!\n");

   while (TRUE)
   {
/* wait for a request */

    msgQReceive(leps_Q, (char *) &psub, 4, WAIT_FOREVER);

    switch ((int) psub->d)
    {

     case CHK_SYS:
     {
         printf("checking system.\n");
         /*if(psub->e==1)*/
           check_sys();

         psub->g=ENABLE_stat;
         if(ENABLE_stat==1) printf("HV enabled.\n");
         check=1;
     }
     break;

     case POWER:
     {
         char   *bon = "on\r\n";
         char   *boff = "off\r\n";
         const char   *pon = "on";
         const char   *pon1 = "ON";
         char    bufout3[70] = "\0";
         char    bufout4[20] = "\0";
         int     pr = strlen(bon);
         int     pt = strlen(boff);
         int     n , i, DMD_sum=0;
         static int first_time=0;

         islot=(int)psub->b;

         if(check==0) check_sys();
         else check=0;
```

```
if (SERIAL_stat != 0 && LeCroy_stat != 0)
 {  if (ENABLE_stat==1)
    {
     psub->g=1;   /* the HV enable light on */
     if(VOLT_stat==1) psub->f=1; /* the HV ON/OFF light on */
     else psub->f=0;            /* the HV ON/OFF light off*/

     if ((int) psub->c == 2 && VOLT_stat==0) /* 2==ON */

       {
         write(fd, bon, pr);
         n = timing(60, bufout3,70);
         if (n != 0)
           {
             bufout3[n]='\0';
             if ( strstr(bufout3+pr,pon) != NULL &&
                   strstr(bufout3+pr,pon1) == NULL)
              { printf("HV on successful\r\n");
                psub->f=1;
               }
            /*  first_time=0; */

           }
      /*   psub->f=1; */
       }
     if ((int) psub->c == 1 && VOLT_stat==1) /*1==OFF */

      {
        for(i=0; i<MAX_CHANS; i++)
             DMD_sum += demand[i];
        if (DMD_sum==0)
        {
          write(fd, buff, pt);
          n = timing(60, bufout4,20);

          if (n != 0)
            {
             bufout4[n]='\0';

             if ( strstr(bufout4+pt,"0>") != NULL)
              { printf("HV off successful.\r\n");
                psub->f=0;
               }

            }
        }
        else
         printf("Cann't turn power off with non-zero demand voltage!\n");


     }
 }
```

```
        else psub->g=0; /* the HV enable light off */
      }

  }
  break;

case READ_I:
{
    int    k, n, i;
    static int count=0;
    char  buff[30]="\0";
    char  *temp;
    const char *value=",";
    const char *star="*";
    const char *flag="S";
    char *temp1="\0", *temp2="\0";
    char  *temp3="\0";
    char Archer[800]="\0";

    islot=(int)psub->b;

    sprintf(buff, "Read (%d,0-15)\r\n",islot);
    write(fd, buff, strlen(buff));
    n = timing(60, bufout_VI, 800);
    /* 16 channels have more chars,so use 800 */

    temp = (char *)strstr(bufout_VI, value);
    strcpy(Archer, temp+1);
    temp=(char *)strstr(Archer, value);
    /* skip the very first comma */
    /* Because there is a comma in the command at the bufout string */


    for (k=0; k<MAX_CHANS; k++)
       {
         if((temp1=(char*)strstr(temp,star))!=NULL){
             trip_stat1[k]=1;
           /*strstr(&bufout_VI[(k+1)*40],star)*/
             }
         if((temp2=(char*)strstr(temp,flag))!=NULL){
             trip_stat2[k]=1;
             }

         for(i=28; i<36; i++)
             temp1[k][i-28]=*(temp+i);
         strcpy(Archer, temp+1);
        /*  temp = (char *)strstr(&bufout_VI[(k+2)*40], value);
          this is for L1444, 8 channels. */
         temp = (char *)strstr(Archer, value);

       }
    psub->e=atof(temp1[0])/100.0; /* there is a factor 100 */
```

```
b->f=atof(tempI[1])/100.0;
b->g=atof(tempI[2])/100.0;
b->h=atof(tempI[3])/100.0;
b->i=atof(tempI[4])/100.0;
b->j=atof(tempI[5])/100.0;
b->k=atof(tempI[6])/100.0;
b->l=atof(tempI[7])/100.0;


EAD_I:

b->e=atof(tempI[8])/100.0;
b->f=atof(tempI[9])/100.0;
b->g=atof(tempI[10])/100.0;
b->h=atof(tempI[11])/100.0;
b->i=atof(tempI[12])/100.0;
b->j=atof(tempI[13])/100.0;
b->k=atof(tempI[14])/100.0;
b->l=atof(tempI[15])/100.0;


EAD_V:

  k, n, i;
 *temp;
t char *value=",";
 Archer[800]="0";

=(int)psub->b;

 = (char *)strstr(bufout_VI, value);
)y(Archer, temp+1);
=(char *)strstr(Archer, value);

k=0; k<MAX_CHANS; k++)

  for(i=18; i<26; i++)
    tempV[k][i-18]=*(temp+i);
*   temp = (char *)strstr(&bufout_VI[(k+2)*40], value);
  this is for L1444, 8 channels. */
    strcpy(Archer, temp+1);
    temp = (char *)strstr(Archer, value);



->e=atof(tempV[0]);
->f=atof(tempV[1]);
->g=atof(tempV[2]);
->h=atof(tempV[3]);
->i=atof(tempV[4]);
->j=atof(tempV[5]);
```

```
            psub->k=atof(tempV[6]);
            psub->l=atof(tempV[7]);

          }
          break;

        case READ_V1:
        {
            psub->e=atof(tempV[8]);
            psub->f=atof(tempV[9]);
            psub->g=atof(tempV[10]);
            psub->h=atof(tempV[11]);
            psub->i=atof(tempV[12]);
            psub->j=atof(tempV[13]);
            psub->k=atof(tempV[14]);
            psub->l=atof(tempV[15]);

        }break;

    /* showing the trip status */

         case TRIP_STAT1:
         {int i;
          psub->e=trip_stat1[0];
          psub->f=trip_stat1[1];
          psub->g=trip_stat1[2];
          psub->h=trip_stat1[3];
          psub->i=trip_stat1[4];
          psub->j=trip_stat1[5];
          psub->k=trip_stat1[6];
          psub->l=trip_stat1[7];
          for(i=0;i<8;i++) trip_stat1[i]=0;

         }break;

         case TRIP1_STAT1:
         {int i;
          psub->e=trip_stat1[8];
          psub->f=trip_stat1[9];
          psub->g=trip_stat1[10];
          psub->h=trip_stat1[11];
          psub->i=trip_stat1[12];
          psub->j=trip_stat1[13];
          psub->k=trip_stat1[14];
          psub->l=trip_stat1[15];
          for(i=8;i<15;i++) trip_stat1[i]=0;

         }break;

        case TRIP_STAT2:
        {int i;
         psub->e=trip_stat2[0];
         psub->f=trip_stat2[1];
         psub->g=trip_stat2[2];
```

```
        psub->h=trip_stat2[3];
        psub->i=trip_stat2[4];
        psub->j=trip_stat2[5];
        psub->k=trip_stat2[6];
        psub->l=trip_stat2[7];
        for(i=0;i<8;i++) trip_stat2[i]=0;

       }break;

       case TRIP1_STAT2:
       {int i;
        psub->e=trip_stat2[8];
        psub->f=trip_stat2[9];
        psub->g=trip_stat2[10];
        psub->h=trip_stat2[11];
        psub->i=trip_stat2[12];
        psub->j=trip_stat2[13];
        psub->k=trip_stat2[14];
        psub->l=trip_stat2[15];
        for(i=8;i<MAX_CHANS;i++) trip_stat2[i]=0;
       }break;

     case SET_VLIM:
     {
      char buff1[30]="\0";
      Vlim_s_i= (int)psub->e;
      Vlim_s_f= (int)psub->f;
      Vlim_c_i= (int)psub->g;
      Vlim_c_f= (int)psub->h;
      Vlim = psub->i;
      if(VOLT_stat==0)
      {
       if(Vlim_s_i<Vlim_s_f && Vlim_c_i< Vlim_c_f)
         {sprintf(buff1, "set voltage limit (%d-%d,%d-%d) %d\r\n", Vlim_s_i, Vlim_s_f, Vlim_c_i,
Vlim_c_f, Vlim);
          write(fd, buff1, strlen(buff1));
         }
       if(Vlim_s_i==Vlim_s_f && Vlim_c_i<Vlim_c_f)
         {sprintf(buff1, "set voltage limit (%d,%d-%d) %d\r\n", Vlim_s_i, Vlim_c_i, Vlim_c_f, Vlim);
          write(fd, buff1, strlen(buff1));
         }
       if(Vlim_s_i<Vlim_s_f && Vlim_c_i==Vlim_c_f)
         {sprintf(buff1, "set voltage limit (%d-%d,%d) %d\r\n", Vlim_s_i, Vlim_s_f, Vlim_c_i,Vlim);
          write(fd, buff1, strlen(buff1));
         }
       if(Vlim_s_i==Vlim_s_f && Vlim_c_i==Vlim_c_f)
         {sprintf(buff1, "set voltage limit (%d,%d) %d\r\n", Vlim_s_i, Vlim_c_i, Vlim);
          write(fd, buff1, strlen(buff1));
         }
       printf("Set vlim ready!\n");
      }


     }break;
```

```
case SET_DEMAND:
{
 int i;
 char  buff1[MAX_CHANS][30]={"\0"};

 islot=(int)psub->b;
demand[0]=psub->e;
demand[1]=psub->f;
demand[2]=psub->g;
demand[3]=psub->h;
demand[4]=psub->i;
demand[5]=psub->j;
demand[6]=psub->k;
demand[7]=psub->l;

for(i=0; i<8; i++){
     if(demand[i]!=demand_old[i]){
      if(polarity==0){
        sprintf(buff1[i],"write (%d,%d) %d\r\n",islot,i,-(int)demand[i]);
     /*   printf("Current Polarity is Negative !\n"); */
        }
      else{
        sprintf(buff1[i],"write (%d,%d) %d\r\n",islot,i,(int)demand[i]);
      /*  printf("Current Polarity is Positive !\n"); */
        }

      write(fd, buff1[i], strlen(buff1[i]));
      demand_old[i]=demand[i];
      }
}


}break;

case SET_DEMAND1:
{
 int i;
 char  buff1[MAX_CHANS][30]={"\0"};

 islot=(int)psub->b;
demand[8]=psub->e;
demand[9]=psub->f;
demand[10]=psub->g;
demand[11]=psub->h;
demand[12]=psub->i;
demand[13]=psub->j;
demand[14]=psub->k;
demand[15]=psub->l;

for(i=8; i<MAX_CHANS; i++){
     if(demand[i]!=demand_old[i]){
      if(polarity==0)
        sprintf(buff1[i],"write (%d,%d) %d\r\n",islot,i,-(int)demand[i]);
```

```
            else
              sprintf(buff1[i],"write (%d,%d) %d\r\n",islot,i,(int)demand[i]);

            write(fd, buff1[i], strlen(buff1[i]));
            demand_old[i]=demand[i];
           }
      }

     }break;

     case SET_RAMP:
     {
       char buff1[30]="\0";

       ramp= (int)psub->i;
       if((int)psub->j!=0)ramp_time=(int)psub->j;
       step=(int)(ramp/ramp_time);
       if(VOLT_stat==0){
        if(step>=10){
          sprintf(buff1, "set ramp (0-%d, 0-15) %d\r\n", L1443, step);
          write(fd, buff1, strlen(buff1));
          }
        else {
          sprintf(buff1, "set ramp (0-%d, 0-15) 10\r\n", L1443);
          write(fd, buff1, strlen(buff1));
          }
        printf("Set ramping rate ready!\n");
       }


     }break;

/* for 1444 modules */

     case SET_ITRIP:
     {
       char buff1[30]="\0";
       Itrip_s_i=(int)psub->e;
       Itrip_s_f=(int)psub->f;
       Itrip_c_i=(int)psub->g;
       Itrip_c_f=(int)psub->h;
       Itrip = 50*(int)psub->i; /* because there is a factor 50 */
       if(VOLT_stat==0)
       {
          if(Itrip_s_i<Itrip_s_f && Itrip_c_i< Itrip_c_f)
           {sprintf(buff1, "set current trip (%d-%d,%d-%d) %d\r\n", Itrip_s_i, Itrip_s_f, Itrip_c_i, Itrip_c_f,
Itrip);
           write(fd, buff1, strlen(buff1));
           }
          if(Itrip_s_i==Itrip_s_f && Itrip_c_i<Itrip_c_f)
           {sprintf(buff1, "set current trip (%d,%d-%d) %d\r\n", Itrip_s_i, Itrip_c_i, Itrip_c_f, Itrip);
           write(fd, buff1, strlen(buff1));
           }
          if(Itrip_s_i<Itrip_s_f && Itrip_c_i==Itrip_c_f)
```

```
        {sprintf(buff1, "set current trip (%d-%d,%d) %d\r\n", Itrip_s_i, Itrip_s_f, Itrip_c_i,Itrip);
         write(fd, buff1, strlen(buff1));
        }
       if(Itrip_s_i==Itrip_s_f && Itrip_c_i==Itrip_c_f)
        {sprintf(buff1, "set current trip (%d,%d) %d\r\n", Itrip_s_i, Itrip_c_i, Itrip);
         write(fd, buff1, strlen(buff1));
        }
       printf("Set itrip ready!\n");
     }

   }break;

   case POLARITY:
   {
    polarity=(int)psub->e; /* default is 0 which means negative */
    printf("Polarity chosed! \n");

   }break;


/*   case SET_AC_TRIP:
   {
    AC_trip_s_i=(int)psub->e;
    AC_trip_s_f=(int)psub->f;
    AC_trip_c_i=(int)psub->g;
    AC_trip_c_f=(int)psub->h;
    AC_trip = (int)psub->i;
   }break;
*/

   case SHOW_LIMITS:
   {
     char bufLIM[30]="\0";
     char bufoutLIM[1200]="\0";
     int n;

     islot=psub->b;
     if(psub->e==1){
        sprintf(bufLIM, "show limits (%d, 0-15)\r\n", islot);
        write(fd, bufLIM, strlen(bufLIM));
        n=timing(60,bufoutLIM,800);
        printf("Show Limits:\n%s\n", bufoutLIM);
       }

   }break;

   case SHOW_AC_TRIP:
   {
     char buf[30]="\0";
     char bufout[800]="\0";
     int n;

     islot=psub->b;
     if(psub->e==1){
```

```
        sprintf(buf, "show ac_trip (%d, 0-15)\r\n", islot);
        write(fd, buf, strlen(buf));
        n=timing(60,bufout,800);
        printf("Show AC trip:\n%s\n", bufout);
       }

}break;

case SHOW_VI:
{    char  buffVI[30]="\0";
     char  bufoutVI[800]="\0";
     int n;
     islot=(int)psub->b;


     if(psub->e==1){
       sprintf(buffVI, "Read (%d,0-15)\r\n",islot);
       write(fd, buffVI, strlen(buffVI));
       n = timing(60, bufoutVI, 800);
       printf("Show Voltage and Current:%s\n", bufoutVI);
      }

}break;

case SHOW_MODULES:
{    char  *bufin = "show modules\r\n";
     char bufout[800]="\0";
     int n;

     islot=(int)psub->b;
     if (psub->e==1){
       write(fd, bufin, strlen(bufin));
       n = timing(60, bufout, 800);
       printf("Current modules are\n%s\n", bufout);
      }

}break;

case SET_CLEAR:
{
   char *buff4="clear\r\n";
   static int clr=0;
   clr=(int)psub->e;

   if(clr==1)
      {write(fd, buff4, strlen(buff4));
       clr=0;
       printf("Cleared all the old settings.\n");
       }

 }break;
}                          /* end of switch */

process the record again to do asynchronous completion */
```

```
        prset = (struct rset *) (psub->rset);
        dbScanLock((struct dbCommon *) psub);
        (*prset->process) (psub);
        dbScanUnlock((struct dbCommon *) psub);
      }                                 /* end while */

    }

    /*
     * Routine to read  in specified time with specified
     * minimum bytes:
     *     Passed:
     *          min: minimum byte count
     *          buffer: buffer address
     *          size: buffer size
     *     Returned:
     *          ZERO: read failed or timed out
     *          >ZERO: actual byte count
     */
    #define FIRST_WAIT 4
    int timing(int wait, char *buffer, int size)
    {
      int    time_left = wait - FIRST_WAIT,
             look_ahead,
             cur_pos = 0,
             buf_rem = size - 1, /* leave room for NULL! */
             tot_read = 0,
             act_read = 0;
      int    status;

      taskDelay(FIRST_WAIT);

      while (time_left-- && buf_rem > 0)
      {
        /* Look ahead in tty buffer */
        status = ioctl(fd, FIONREAD, (int) &look_ahead);
        if (status == ERROR) break;
        /* Read whatever is there, which might be more than
          ioctl...but keep within buffer */
        if (look_ahead > 0)
        {
          act_read = read(fd, &buffer[cur_pos], buf_rem);
          cur_pos += act_read;
          tot_read += act_read;
          buf_rem -= act_read;
          /*
           *  Terminate, check if done by looking for 'prompt' string
           *  at end of buffer.
           */
          buffer[cur_pos] = '\0';
          if (cur_pos > 3 && strcmp(buffer + cur_pos - 3, "0> ") == NULL)
          {
    #ifdef DEBUG
```

```
            printf("Got (%s)\n", buffer);
            printf("Took %d\n", (wait_time_left-1) );
#endif
            return (tot_read);
          }
        }
        /* Wait and retry if nothing, or not enough */
        taskDelay(1);
      }

    return (0);
}

void check_sys()
{           int    n;
            static int inode=0;
            char   *crop="\0";
            const char *ss = "LeCroy 1445A";
            const char *es = "Power supply disabled";
            const char *hv = "high voltage";
            const char *poff = "off";
            char *lim="show limits (0, 0-15)\r\n";
            char   bufout1[200] = "\0";
            char   *bufin1 = "mainframe 0\r\n";
            int    pr = strlen(bufin1);
            char   bufout2[400] = "\0";
            char   *bufin2 = "show status\r\n";
            int    pt = strlen(bufin2);
            char   bufout3[400]= "\0";

            /* check line */


             write(fd, bufin1, pr);
             n=timing(60,bufout1,200);
        /*     printf ("n=%d\n", n); */
             if ( n == 0 )
                SERIAL_stat = 0;
             else
             {
              bufout1[n]='\0';
              if ((char *)strstr(&bufout1[pr], "0>") != NULL)
                 SERIAL_stat = 1;

              else
                 SERIAL_stat = 0;
             }

/*            psub->a = SERIAL_stat; */


            /* check status */
            if (SERIAL_stat == 0)
              LeCroy_stat = 0;
```

```
            else
            {
               write(fd, bufin2, pt);
               n=timing(60,bufout2,400);
               printf ("n=%d\n", n);

               printf("bufout2=\n%s\n", bufout2);

               if ( n == 0 )
                  LeCroy_stat = 0;
               else
                {
                  bufout2[n]='\0';
                  if ((cmp = (char *)strstr(&bufout2[pt], ss)) != NULL)
                        LeCroy_stat = 1;
                  else
                        LeCroy_stat = 0;
                }
        /*      write(fd, lim, strlen(lim));
               n=timing(60,bufout3,400);
               printf("bufout3=\n%s\n",bufout3); */
            }
/*          psub->b = LeCroy_stat; */
/* check  enable*/
            if (LeCroy_stat == 0)
               ENABLE_stat = 0;
            else
            {
             if ( strstr(cmp,es) != NULL)
                ENABLE_stat = 0;
             else
                ENABLE_stat = 1;

            }
/*          psub->c = ENABLE_stat; */


            /* check  power on/off */
            if (LeCroy_stat == 0)
               VOLT_stat = 0;
            else
            {
             if ( strstr(cmp,poff) != NULL)

                VOLT_stat = 0;
             else
                VOLT_stat = 1;
            }
}
```

Appendix 5 SNL source code for Wavetek350

```
program ppulser
/* vxWorks includes */
%%#include      <vxWorks.h>
%%#include      <string.h>
%%#include      <stdlib.h>
%%#include      <stddef.h>
%%#include      <types.h>
/*%%#include      <dbDefs.h>
 %%#include     <dbAccess.h>
%%#include      <dbCommon.h>*/
%%#include      <stdioLib.h>
%%#include      <msgQLib.h>
%%#include      <ioLib.h>
%%#include      <logLib.h>
%%#include      <stdio.h>
%%#include      <kernelLib.h>
%%#include      <taskLib.h>
%%#include      <tyLib.h>
%%#define  SIZE    7000
%%#define  ADDOFFSET  2048
short main_out;
assign main_out to "TPC_ppulser_MainOut";
monitor main_out;

short main_check;
assign main_check to "TPC_ppulser_MainCheck";
monitor main_check;

short main_out_calc;
assign main_out_calc to "TPC_ppulser_Maincalc";
/* monitor main_out_calc; */


short sync_out;
assign sync_out to "TPC_ppulser_SYNC";
/*monitor sync_out;*/

short out_y;
assign out_y to "TPC_ppulser_WaveOutput";

short out_proc;
assign out_proc to "TPC_ppulser_WaveOutput.PROC";

short slt;
assign slt to "TPC_ppulser_select";
monitor slt;

short wave_input;
assign wave_input to "TPC_ppulser_Input";
monitor wave_input;
```

```
short wave_check;
assign wave_check to "TPC_ppulser_WaveCheck";
monitor wave_check;

short WaveShow;
assign WaveShow to "TPC_ppulser_Show";
monitor WaveShow;

short P_chk;
assign P_chk to "TPC_ppulser_PNT_CHK.PROC";
monitor P_chk;

short P_index;
assign P_index to "TPC_ppulser_PNT_CHK.VAL";
monitor P_index;

short P_value;
assign P_value to "TPC_ppulser_PNT_CHK.A";

short trace_query;
assign trace_query to "TPC_ppulser_TraceQuery";

short TrgMode;
assign TrgMode to "TPC_ppulser_TrgMode";
monitor TrgMode;

short TrgMode_CHK;
assign TrgMode_CHK to "TPC_ppulser_ModeCheck";
monitor TrgMode_CHK;

short FreqMode;
assign FreqMode to "TPC_ppulser_FreqMode"; /* trace frequency mode*/
monitor FreqMode;

short TraceMode_CHK;
assign TraceMode_CHK to "TPC_ppulser_TraceMode";
monitor TraceMode_CHK;

short x;
assign x to "TPC_ppulser_X";

short x_proc;
assign x_proc to "TPC_ppulser_X.PROC";

short y;
assign y to "TPC_ppulser_Y";

short y_proc;
assign y_proc to "TPC_ppulser_Y.PROC";

/*char display[200];*/
string display[600];
assign display to "test2.VAL";
```

```
string screen[500];
assign screen to "TPC_ppulser_New_Trace.VAL";
/* assign screen to "test3.val"; */
monitor screen;

short NewTrace;
assign NewTrace to "TPC_ppulser_New";
monitor NewTrace;

short TraceSize;
assign TraceSize to "TPC_ppulser_Trace_Size";
monitor TraceSize;

short FreqCW;
assign FreqCW to "TPC_ppulser_FreqCW";
monitor FreqCW;

long FreqCW_proc;
assign FreqCW_proc to "TPC_ppulser_FreqCW.PROC";

long FreqRAST;
assign FreqRAST to "TPC_ppulser_FreqRAST";
monitor FreqRAST;

short FreqRAST_proc;
assign FreqRAST_proc to "TPC_ppulser_FreqRAST.PROC";

short RAST; /* always set the RAST mode with freq 1e8 */
assign RAST to "TPC_ppulser_RAST";
monitor RAST;

float Volt;
assign Volt to "TPC_ppulser_Volt";
monitor Volt;

float V;
assign V to "TPC_ppulser_V_readback";

short V_proc;
assign V_proc to "TPC_ppulser_V_readback.PROC";

short V_exceed;
assign V_exceed to "TPC_ppulser_V_exceed";

short V_exceed_proc;
assign V_exceed_proc to "TPC_ppulser_V_exceed.PROC";

float Volt_Offset;
assign Volt_Offset to "TPC_ppulser_V_OFF";
monitor Volt_Offset;

short Volt_Switch;
assign Volt_Switch to "TPC_ppulser_V_SW";
monitor Volt_Switch;
```

```
short wavetek_reset;
assign wavetek_reset to "TPC_ppulser_reset";
monitor wavetek_reset;

short stop;
assign stop to "TPC_ppulser_stop";
monitor stop;

int idle;
int fd;
int status;
int status1;
int n;
%%int CONT=1e8;
%%int Old=0;


/* %%char *file="/tyCo/1"; /tyCo is a VxWorks physical serial channel, but only /tyCo/1 works! Wierd. */

%%char *file='/tyCo/1';
%%char *bufMainOn =":OUTP 1\r\n";
%%char *bufMainOff =":OUTP 0\r\n";
%%char *bufMainStat =":OUTP?\r\n";
%%char *bufSyncOn=":OUTP:SYNC 1\r\n";
%%char *bufSyncOff=":OUTP:SYNC 0\r\n";
%%FILE *Fwfdata; /* waveform data */
%%char *path = "/home/sunstar2/jlin/epics_app/PPulserApp/src/formac.dat";
%%char *input=":TRAC:DATA:POIN WaveTek,";
%%char *input1=":TRAC:DATA:POIN WaveTek1,";
%%char *input2=":TRAC:DATA:POIN WaveTek2,";
%%char *input3=":TRAC:DATA:POIN WaveTek3,";

%%char *inputtest=":TRAC:DATA:POIN TEST,";
%%char *tracename=":TRAC:DEF WaveTek, 7000\r\n";
%%char *tracename1=":TRAC:DEF WaveTek1, 7000\r\n";
%%char *tracename2=":TRAC:DEF WaveTek2, 7000\r\n";
%%char *tracename3=":TRAC:DEF WaveTek3, 7000\r\n";
%%char *name="WaveTek";

%%char *tracetest=":TRAC:DEF TEST1, 14\r\n";
%%char *wavequery=":TRAC:DATA:POIN? WaveTek,";
%%char *wavequery1=":TRAC:DATA:POIN? WaveTek1,";
%%char *wavequery2=":TRAC:DATA:POIN? WaveTek2,";
%%char *wavequery3=":TRAC:DATA:POIN? WaveTek3,";

%%char *wavetest=":TRAC:DATA:POIN? TEST,";
%%char *tracequery=":TRAC:CAT?\r\n";
%%char *trigger_ext=":TRIG:SOUR:STAR EXT\r\n";
%%char *trigger_int=":TRIG:SOUR:STAR INT\r\n";
%%char *TrgModeCheck=":TRIG:SOUR:STAR?\r\n";
%%char *TraceModeCheck=":TRAC:MODE?\r\n";
%%char *TraceMode_CW=":TRAC:MODE CW\r\n";
%%char *TraceMode_RAST=":TRAC:MODE RAST\r\n";
```

```
%%char *Rst="*RST\r\n";
%%char *Freq_RAST=":FREQ:RAST";
%%char *Freq_CW=":FREQ:CW";
%%char *Voltage_Offset=":VOLT:OFFS";
%%char *Voltage=":VOLT:AMPL";
%%char *ModeRAST="RAST";
%%char *ModeCW="CW";
%%char *CWMode="Don't need CW mode now.";

%%char *input1;
%%char temp[100]="\0";
%%char temp1[100]="\0";
%%char temp2[100]="\0";
%%char temp3[100]="\0";
%%char bufoutTQ[200]="\0";
%%char bufoutMQ[200]="\0";
%%char bufout_check[200]="\0";
%%char bufoutTMC[200]="\0";
%%char bufoutTrace[200]="\0";
%%char bufout[200]="\0";
%%char *off="0";
%%char *on="1";


int lenMainOn;
int lenMainOff;
int lenMainStat;
int lenSyncOn;
int lenSyncOff;
int length;
int lenName;
int lenInput;
int lenTquery;
int i=0;
int wavedata[7200];
int wavedata_out[7200];
int look_ahead;
int cur_pos;

ss define_trace
{
        state start
        {
                when(NewTrace==1) {
                  sprintf(temp, ":TRAC:DEF %s, %d\r\n",screen,TraceSize);
                  length=strlen(temp);
                  write(fd,temp,length);
                  } state next
        }

        state next
        {
                when(NewTrace==0) {
                  sprintf(display, "define a new trace %s",screen);
```

```
                pvPut(display);
                } state start
        }
}

ss main_out{

        state init
        {
                when(delay(0.2)) {

                        fd = -1;
                        fd = open (file, O_RDWR, 0644); /*read and write. */
                        if (fd == ERROR)
                        {
                        %% printf("Cann't open %s\n", file);
                         exit (-1);
                        }

                        status = ioctl(fd, FIOSETOPTIONS, OPT_RAW);
              if (status == ERROR)
                if(status==0)
                  exit (-1);
                else
                %% printf("serial port opened OK!\n");
               /*  strcpy(display,bufMainOn);
                pvPut(display);*/


                }state main_on

        }

        state main_on
        {
                when(main_out == 1){

              lenMainOn=strlen(bufMainOn);
              write(fd, bufMainOn, lenMainOn);
              printf("Turn the mainout on!\n");
              main_out_calc=1;
              pvPut(main_out_calc);

             /*  fdprintf(fd, "OUPT 1\r\n"); fdprintf doesn't work here*/
                }state main_off
        }

        state main_off
        {
                when(main_out == 0) {

          lenMainOff=strlen(bufMainOff);
          write(fd, bufMainOff, lenMainOff);
          printf("Turn the mainout off!\n");
```

```
            main_out_calc=0;
            pvPut(main_out_calc);
                ]state main_on

        /*      when(main_out ==0) {
                printf("no actions on mainout. \n");
                } state main_check */
        }
}

ss main_status
{
        state main_check
        {
                when(main_check==1) {
                 status1 = ioctl(fd, FIOFLUSH, 0);/*clear the i/o buffer. */
                 lenMainStat=strlen(bufMainStat);
                 write(fd, bufMainStat,lenMainStat);
                 n=timing(60,bufoutMQ,200);
                 printf("mainout status is: \n'%s\n",bufoutMQ);

                 if(strstr(bufoutMQ,on)!=NULL)
                  { main_out_calc=1;
                    pvPut(main_out_calc);
                    printf("mainout is currently on!\n");
                    sprintf(display,"mainout is %s!",on);
                    pvPut(display);

                   }
                 if(strstr(bufoutMQ,off)!=NULL)
                  { main_out_calc=0;
                    pvPut(main_out_calc);
                    printf("mainout is currently off!\n");
                    sprintf(display,"mainout is %s!",off);
                    pvPut(display);

                    }

                  }state main_check_1
        }

        state main_check_1
        {
                when(main_check==0) {
                printf("check mainout status end. \n");
                }state main_check
        }

}

ss syncronization
{
        state init
        {
```

```
                when()
                {
                pvMonitor(sync_out);
                } state sync_on
        }

    state sync_on
    {
        when(sync_out==1){
         lenSyncOn=strlen(bufSyncOn);
         write(fd,bufSyncOn,lenSyncOn);
         printf("Turn SYNC on!\n");
         } state sync_off
    }

    state sync_off
    {
        when(sync_out==0){
          lenSyncOff=strlen(bufSyncOff);
          write(fd,bufSyncOff,lenSyncOff);
          printf("Turn SYNC off!\n");
          } state sync_on
    }

}

ss trace_query
{
        state init
        {
                when(){
                pvMonitor(trace_query);
                } state send_query
        }

        state send_query
        {
                when(trace_query==1){
                 status1 = ioctl(fd, FIOFLUSH, 0);
                 lenTquery=strlen(tracequery);
                 write(fd,tracequery,lenTquery);
                 printf("sending query. \n");


                 n=timing(60,bufoutTQ,100);

                 printf("n=%d\n current traces are: \n%s\n", n,bufoutTQ);
                 sprintf(display,"current trace %s",bufoutTQ);
                /*  %%strcpy(display, bufoutTQ);*/
                 pvPut(display);

                 } state end_query
        }
        state end_query
```

```
        {
                when(trace_query==0) {
                printf("query end. \n");
                } state send_query
        }
}

ss wave_sel
{
        state start
        {
                when(slt==1)
                {
            sprintf(temp,":TRAC:SEL %s\r\n",name);
            length=strlen(temp);
            write(fd, temp,length);
         } state next
    }

    state next
    {
        when(slt==0)
        {
            sprintf(display, "Trace '%s' selected!",name);
            pvPut(display);
         } state start
    }
}


ss wave_data
{
        state init
        {
                when(){
            Fwfdata=fopen(path,"r");
            if(Fwfdata!=NULL)
              {for(i=0;i<SIZE;i++)
               { fscanf(Fwfdata,"%d\n",&wavedata[i]);
               /*  printf("wavedata[%d]=%d\n",i,wavedata[i]);*/
               /*  delay(2000); */
               }
              printf("file open success.\n");

              }
            else printf("file open failed. \n");
            fclose(Fwfdata);
           /*  status1 = ioctl(fd, FIOFLUSH, 0); */
            lenName=strlen(tracename);
            write(fd,tracename,lenName);
            printf("Define a new trace %s\n", tracename);
            lenName=strlen(tracetest);
            write(fd,tracetest,lenName);
            printf("Define a new trace %s\n", tracetest);
```

```
                } state wave_input
        }

        state wave_input
        {
                when(wave_input==1){
                        /*for(i=0;i<SIZE;i++)*/
                        i=0;
                        while (stop==0 && i<SIZE)
                        {
              /*  wave_I_data=wavedata[i];
                pvPut(wave_I_data);
                data_proc=1;
                pvPut(data_proc);*/
                         x=i;
                         pvPut(x);
                         x_proc=1;
                         pvPut(x_proc);
                         y=wavedata[i];
                         pvPut(y);
                         y_proc=1;
                         pvPut(y_proc);
                %%sprintf(temp,"%d,%d\r\n",i,wavedata[i]+ADDOFFSET);
                         %%strcpy(temp3,input);
                        /* %%strcpy(temp3,inputtest);*/
                         %%strcat(temp3,temp);
                        /* printf("input= %s\n",temp3); */
                         status1 = ioctl(fd, FIOFLUSH, 0);
                         lenInput=strlen(temp3);
                         write(fd,temp3,lenInput);
                        /* delay(2000); */
                        /* taskDelay(sysClkRateGet()/4);*/
                         taskDelay(sysClkRateGet()/50);
                         i++;
                         }
                } state   wave_input1
        }

        state wave_input1
        {
                when(wave_input==0) {
                printf("wave input finished. \n");
                } state wave_input
        }
}

ss wave_show
{
        state start
        {
                when(WaveShow==1) {
                 /* for (i=0;i<SIZE;i++)
                   {
```

```
                        x=i;
                        pvPut(x);
                        x_proc=1;
                        pvPut(x_proc);
                        y=wavedata[i];
                        pvPut(y);
                        y_proc=1;
                        pvPut(y_proc);
                        taskDelay(sysClkRateGet()/100);
                    }
                    */
                    i=0;
                    strcpy(display,"Showing the wave data.....");
                    pvPut(display);
                    while(stop==0 && i<SIZE){
                        x=i;
                        pvPut(x);
                        x_proc=1;
                        pvPut(x_proc);
                        y=wavedata[i];
                        pvPut(y);
                        y_proc=1;
                        pvPut(y_proc);
                        taskDelay(sysClkRateGet()/50);
                        i++;
                    }
                } state next
            }

            state next
            {
                when(WaveShow==0) {
                 strcpy(display, "Showing wave data finished. ");
                 pvPut(display);
                 } state start
            }
}


ss wave_check
{
        state check_start
        {
                when(wave_check==1) {
                /* status1 = ioctl(fd, FIOFLUSH, 0);*/

                 strcpy(display,"Checking trace data .....");
                 pvPut(display);

                /* for(i=0;i<SIZE;i++)*/
                 i=0;
                 while(stop==0 && i<SIZE)
                 {
```

```
                %%sprintf(temp1,"%d\r\n",i);
                %%strcpy(temp2,wavequery);
                /* %%strcpy(temp2,wavetest);*/
                %%strcat(temp2,temp1);
                status1 = ioctl(fd, FIOFLUSH, 0);
                length=strlen(temp2);
                write(fd,temp2,length);
                n=timing(60,bufout_check,200);
                /*printf("checking the inputted data:\n%s\n",bufout_check);*/
                out_y=atoi(bufout_check)%2048; /* wierd RS232 output*/
                pvPut(out_y);
                wavedata_out[i]=out_y;
                out_proc=1;
                pvPut(out_proc);
                x=i;
                pvPut(x);
                x_proc=1;
                pvPut(x_proc);
                /* printf("number %d=%d\n",i,out_y);*/
                /* %% bufout_check[0]='\0'; */
                /* taskDelay(sysClkRateGet()/4); */
                taskDelay(sysClkRateGet()/50);
                i++;
                }
              }state check_stop
        }

        state check_stop
        {
                when(wave_check==0) {
                printf("check trace data finished. \n");
                strcpy(display,"Check trace data finished.");
                pvPut(display);
                } state check_start
        }
}

ss point_check
{
        state start
        {
                when(P_index!=Old){
                  sprintf(temp1,"%d\r\n",P_index);
                  strcpy(temp2,wavequery);
                 /*strcpy(temp2,wavetest);*/
                  strcat(temp2,temp1);
                  status1 = ioctl(fd, FIOFLUSH, 0);
                  length=strlen(temp2);
                  write(fd,temp2,length);
                  n=timing(60,bufout_check,200);
                  wavedata_out[P_index]=atoi(bufout_check)%2048; /* wierd RS232 output*/
                  P_value=wavedata_out[P_index];
                  pvPut(P_value);
                  sprintf(display, "check point #%d with value %d", P_index,P_value);
```

```
                    pvPut(display);
                    taskDelay(sysClkRateGet()/16);
                    P_chk=1;
                    pvPut(P_chk);
                    Old=P_index;

                    } state start
        }
/*
        state next
        {
                when(P_chk==0){
                  idle=1;
                } state start
        }
*/
}

ss WaveTrigger
{
        state select_mode
        {
                when(TrgMode==0) {
                 length=strlen(trigger_int);
                 write(fd,trigger_int,length);
                 printf("Internal trigger mode selected. \n");
                 } state select_mode1
        }

        state select_mode1
        {
                when(TrgMode==1) {
                 length=strlen(trigger_ext);
                 write(fd,trigger_ext,length);
                 printf("External trigger mode selected. \n");

                 } state select_mode
        }
}

ss trgmode_check
{
        state mode_check
        {
                when(TrgMode_CHK==1){
                 status1 = ioctl(fd, FIOFLUSH, 0);
                 length=strlen(TrgModeCheck);
                 write(fd,TrgModeCheck,length);
                 n=timing(60,bufoutTMC,200);
                 printf("Current trigger mode=%s\n",bufoutTMC);
                 /*%%strcpy(display, bufoutTMC);*/
                 sprintf(display,"Trigger mode:%s",bufoutTMC);
                 pvPut(display);
```

```
                } state mode_check1
        }

        state mode_check1
        {
                when(TrgMode_CHK==0) {
                printf("check trigger mode finished. \n");
                } state mode_check
        }
}

ss WaveFreq
{
        state select_mode
        {
                when(FreqMode==0) {
                length=strlen(TraceMode_RAST);
                write(fd,TraceMode_RAST,length);
                sprintf(temp," %d\r\n",FreqRAST);
                strcpy(temp1,Freq_RAST);
                strcat(temp1,temp);
                length=strlen(temp1);
                write(fd,temp1,length);
                sprintf(display, "RAST frequency is %d.\n",FreqRAST);
                pvPut(display);
                } state select_mode1
        }

        state select_mode1
        {
                when(FreqMode==1) {
                /*  length=strlen(TraceMode_CW);
                write(fd,TraceMode_CW,length);
                sprintf(temp," %d\r\n",FreqCW);
                strcpy(temp1,Freq_CW);
                strcat(temp1,temp);
                length=strlen(temp1);
                write(fd,temp1,length);
                printf("CW frequency is %d.\n",FreqCW);
                */
                printf("CW frequency not needed now. \n");

                /* %%strcpy(display, CWMode);
                pvPut(display);*/

                } state select_mode
        }

}

ss DefRAST
{
        state start
        {
```

```
                when(RAST==1) {
                 length=strlen(TraceMode_RAST);
                 write(fd,TraceMode_RAST,length);
                 sprintf(temp," %d\r\n",CONT);
                 strcpy(temp1,Freq_RAST);
                 strcat(temp1,temp);
                 length=strlen(temp1);
                 write(fd,temp1,length);
                 printf("RAST frequency is set to default %d.\n",CONT);
                 FreqRAST=CONT;
                 pvPut(FreqRAST);
                 FreqRAST_proc=1;
                 pvPut(FreqRAST_proc); /* change the input RAST box to be CONT. */
                 } state next
        }

        state next
        {
                when(RAST==0) {
                 printf("Set the trace mode to RAST 1e8.\n");

                } state start
        }
}

ss Tracemode_check
{
        state mode_check
        {
                when(TraceMode_CHK==1){
                 status1 = ioctl(fd, FIOFLUSH, 0);
                 length=strlen(TraceModeCheck);/*check the trace mode */
                 write(fd,TraceModeCheck,length);
                 n=timing(60,bufoutTrace,200);
                 printf("Current Frequency mode=%s\n",bufoutTrace);
                 if(strstr(bufoutTrace,ModeRAST)!=NULL)
                   { strcpy(temp,Freq_RAST);
                    strcat(temp,"?\r\n");
                    length=strlen(temp);
                    write(fd,temp,length);
                    n=timing(60,bufout,200);
                    printf("RAST mode frequency is %s\n",bufout);
                   }
                 if(strstr(bufoutTrace,ModeCW)!=NULL)
                   { strcpy(temp,Freq_CW);
                    strcat(temp,"?\r\n");
                    length=strlen(temp);
                    write(fd,temp,length);
                    n=timing(60,bufout,200);
                    printf("CW mode frequency is %s\n",bufout);
                   }
                 %%strcpy(display, bufout);
                 pvPut(display);
```

```
                } state mode_check1
        }

        state mode_check1
        {
                when(TraceMode_CHK==0) {
                printf("Check Frequency mode finished. \n");
                } state mode_check
        }
}

ss voltage
{
        state start
        {
                when(Volt_Switch==1) {
                  if(Volt<=1.25) {
                    sprintf(temp2," %f\r\n",Volt);
                    strcpy(temp3,Voltage);
                    strcat(temp3,temp2);
                    length=strlen(temp3);
                    write(fd,temp3,length);
                    printf("The voltage is %f\n",Volt);
                    V_exceed=0;
                    pvPut(V_exceed);
                    V_exceed_proc=1;
                    pvPut(V_exceed_proc);

                  }
                  else {
                    printf("voltage exceeds the 1.25v limit. \n");
                    sprintf(display,"voltage %f exceeds the 1.25v limit.",Volt);
                    pvPut(display);
                    V_exceed=1;
                    pvPut(V_exceed);
                    V_exceed_proc=1;
                    pvPut(V_exceed_proc);
                    }
                  if(Volt_Offset==0) {
                    sprintf(temp2," %f\r\n",Volt_Offset);
                    strcpy(temp3,Voltage_Offset);
                    strcat(temp3,temp2);
                    length=strlen(temp3);
                    write(fd,temp3,length);
                    printf("The voltage offset is %f\n",Volt_Offset);
                  }
                } state next
        }
        state next
        {
                when(Volt_Switch==3){
                  if(V_exceed==0){
                    sprintf(display, "voltage set to %f, offset is %f",Volt,Volt_Offset);
                    pvPut(display);
```

```
                    }
                    } state start
        }
}

ss voltage_chk
{
        state start
        {
                when(Volt_Switch==0) {
                  status1 = ioctl(fd, FIOFLUSH, 0);
                  strcpy(temp2,Voltage);
                  strcat(temp2, "?\r\n");
                  length=strlen(temp2);
                  write(fd,temp2,length);
                /*  printf("temp2=%s\n",temp2);*/
                  n=timing(60,bufout,200);
                  printf("After check,\nthe voltage is set to be%s\n",bufout);
                  V=atof(bufout);
                  pvPut(V);
                  V_proc=1;
                  pvPut(V_proc);
                  strcpy(temp2,Voltage_Offset);
                  strcat(temp2,"?\r\n");
                  length=strlen(temp2);
                  write(fd,temp2,length);
                  n=timing(60,bufout,200);
                  printf("After check,\nthe voltage offset is%s\n",bufout);
                } state next
        }

        state next
        {
                when(Volt_Switch==2) {
                  sprintf(display,"Checking:voltage=%f, offset=%f",V,bufout);
                  pvPut(display);
                  } state start
        }
}

ss reset_WaveTek
{
        state Reset
        {
                when(wavetek_reset==1) {
                  length=strlen(Rst);
                  write(fd,Rst,length);
                 } state done
        }

        state done
        {
                when(wavetek_reset==0) {
                  strcpy(display,"Reset the WaveTek!");
```

```
                pvPut(display);
              } state Reset
        }
}


%{
/*
 * Routine to read  in specified time with specified
 * minimum bytes:
 *    Passed:
 *        min: minimum byte count
 *        buffer: buffer address
 *        size: buffer size
 *    Returned:
 *        ZERO: read failed or timed out
 *        >ZERO: actual byte count
 */
#define FIRST_WAIT 4
int timing(int wait, char *buffer, int size)
{
  int   time_left = wait - FIRST_WAIT,
        look_ahead=0,
        cur_pos = 0,
        buf_rem = size - 1, /* leave room for NULL! */
        act_read = 0;
  int   status;
  int   len1=0;
  int   len2=0;
  int   length=0;
  int   k;
  char   *pt="\0";
  char   *sign="\r\n";
  char   *sign1="Enter command >";
  char   archer[800]="\0";
  char   temper[800]="\0";
  char   *temp;

  taskDelay(FIRST_WAIT);
  while(time_left-- && buf_rem>0 )
  {
    /* Look ahead in tty buffer */
    status = ioctl(fd, FIONREAD, (int)&look_ahead);
    if (status == ERROR) exit(-1);
    /* Read whatever is there, which might be more than
      ioctl...but keep within buffer */
    if (look_ahead > 0)
    {
      act_read = read(fd, &buffer[cur_pos], buf_rem);
      cur_pos += act_read;
      buf_rem -= act_read;

     /* buffer[cur_pos] = '\0';*/
      buffer[cur_pos-2]='\0';
```

```
        pt=strstr(buffer, sign);
        return(act_read);

        if(pt!=NULL)
        {  strcpy(archer,pt+1);
           len1=strlen(archer);
           printf("archer string=\n%s\n",archer);
        }
        /* if ((temp=(char *)strstr(archer, sign1)) != NULL)*/

        if ((temp=(char *)strstr(archer, sign)) != NULL)

        /* there is a '\r\n' right before 'Enter Command>'
        and in default setting of RS232 on Wavetek, the echo is off,
        that means there will be no echo 'Enter Command>'
        if the remote setting on the Wavetek is 'ECHO OFF", the we
        can skip the lines below. */
        {
                strcpy(temper,temp);
          /*    printf("temper string=\n%s\n",temper); */
                len2=strlen(temper);
                length=len1-len2+1;
                for(k=1;k<length;k++)
                  buffer[k-1]=*(pt+k);
                buffer[length-1]='\0';
           return (length);
         }

      }
      /* Wait and retry if nothing, or not enough */
      taskDelay(1);
   }
   return (0);
}

}%
```

References

[1] J. W. Harris *et al.* "The STAR Experiment at Relativistic Heavy Ion Collider", Nucl. Phys. A**566** 277c, 1994.

[2] http://www.star.bnl.gov/afs/rhic/star/doc/www/viewgraphs/intro2star/page7.html

[3] B. Muller, Rep. Prog. Phys. **58** 619(1995)

[4] J. Harris and B. Muller, Ann. rev. Nucl. Part. Sci. **46** 71(1996)

[5] K. Eskola and X.N. Wang , Phys. Rev. **D49** 1284 (1994)

[6] K. Geiger, R. Longacre and D.K.Scrivasta, Comput. Phys. Commun. **104** 70 (1997)

[7] S. Vance, M. Gyulassy and X.N. Wang, Proceeding for Quark Matter 97

[8] R.Bellwied, Nucl. Part. Phys. **25** 445 (1999)

[9] C. Wong, Introduction to High-Energy Heavy-Ion Collisions. World Scientific Publishing Co., Singapore (1994)

[10] J.D. Bjorken, Phys. Rev. **D 27** 140 (1983)

[11] K. Geiger and J.I. Kapusta, Phys. Rev. **D 47** 4905 (1993)

[12] S. A. Bass, M. Gyulassy, H. Stocker and W. Greiner, Nucl. Part. Phys. **25** R1-R57 (1999)

[13] T. Blum et al , Phy. Rev., D51, 5153 (1995)

[14] H. Stocker and W. Wreiner, Phys. Rev. **137** 277 (1986)

[15] H. Stocker and B. Muller, LBL-Preprint 12471, unpublished

[16] K. Geiger Phys. Rev. C **48** 4129 (1993)

[17] B. Zhang, M. Gyulassy and Y. Pang, Heavy Ion Phys. **4** 361-8 (1996)

[18] F. Karsch, M.T. Mehr and H.Satz, Z. Phys. C **37** 2469 (1988)

[19] D Blaschke, Nucl. Phys. A **525** 269c (1991)

[20] L. Ramello et al, Proceedings of the Thirteenth International Conference on Ultra-Relativistic Nucleus-Nucleus Collisions-Quark Matter '97, to be published in Nucl. Phys. A (1998)

[21] P. Koch, U. Heinz and J. Pisut, Phys. Lett. **243B** 149; Z. Phys. C **47** 477 (1990)

[22] M.H. Thomas and M. Gyulassy, Nucl. Phys. B **351** 491 (1991)

E. Braaten and M.H. Thomasm Phys. Rev. D **44** R2625 (1991)

[23] A.B. Migdal, Sov. Phys. -JETP **5** 527 (1957)

[24] M. Gyulassy and X.N. Wang, Nucl. Phys. B **420** 583 (1994)

[25] M. Gyulassy, M. Plumer, H. Thoma and X.N. Wang, Nucl. Phys. A **538** 37c (1992)

[26] P.V. Ruuskanen, Nucl. Phys. A 525, 255c (1991)

[27] K. Geiger and J.I. Kapusta, Phys. Rev. Lett. **70**, 1920 (1993)

[28] M.T. Strickland, Phys. Lett. B **331**, 245 (1994)

[29] R.Vogt, B.V. Jacak, P.L. McGaughey and P.V. Ruusmanen, Phys. Rev.D**49** 3345 (1994)

[30] B. Muller and X.N. Wang, Phys. Rev. Lett. **68** 2437 (1992)

[31] J. Kapusta, P. Lichard and D. Seibert, Phys. Rev. D **44** 2774 (1991)

[32] Q. Jones, "Physics Using the Solenoidal Tracker At RHIC (STAR): Development of an Online Data Transfer Link Between the Relativistic Heavy Ion Collider (RHIC) and STAR Slow Controls Using CDEV.", Creighton University preprint CU-PHY-NP 98/01, October, 1998.

[33] http://rsgi01.rhic.bnl.gov/star/starlib/doc/www/html/tpc_l/hard/tperings/page3.html

[34] http://www.atdiv.lanl.gov/aot8/epics/EPICS_description.html

[35] VxWorks Programmer's Guide 5.1, Wind River Systems, Inc., Alameda, CA 1993.