SCTDAQ-
a brief summary

John Hill

University of Cambridge

- Developed for readout of a “few” modules (up to ~20) in institute labs, PS irradiations, systemtest and (from 2000) Test Beam.

- Designed to be modular...
  - libraries for VME modules developed for general-purpose use
  - these libraries also independent of each other

- ...and portable
  - code in ANSI-C (for libraries) and C++ (for ROOT macros)
  - ROOT for menus and histogram presentation is available on all likely platforms

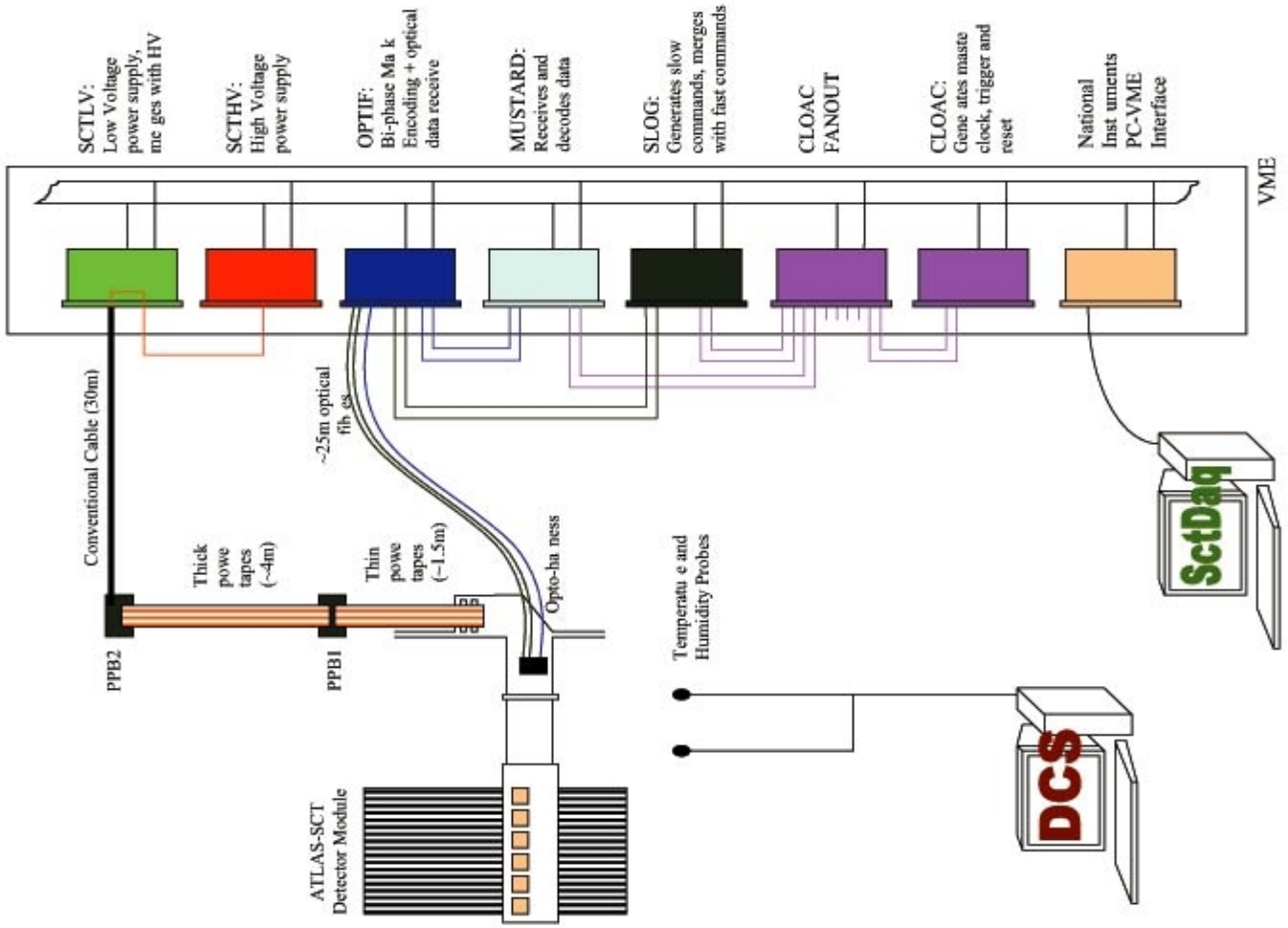
- Main authors: Lars Eklund, Gareth Moorhead, Peter Phillips and John Hill

- Main features of a typical SCTDAQ configuration
  - Windows 95, 98 or NT
  - NI-VXI interface (though BIT3 in also in use)
  - Visual C++ (needed to compile the software)
  - VME Hardware readout modules (all 6U modules)
    - CLOAC (UCL) - Master Clock and Control
    - SLOG (RAL) - Slow Commands to ABCD chips
    - MuSTARD - (RAL/Cambridge) - readout of data from ABCDs
    - SCTLV (Prague) - Low Voltage power supply, and Hard Reset/SELECT signals to front ends.
    - [SCTHV (Krakow) - High Voltage power supply]
  - for optical rather than electrical readout, a BiLED (Oxford) or OpTIF (Cambridge) VME module is required.

- One set of MuSTARD, SLOG, CLOAC modules can read out 6 double-sided SCT detector modules. However, the software is designed to accommodate multiple readout modules, so >6 detector modules can be handled.
- However, readout density is low (a VME crate would be filled if reading out 12 detector modules), so the hardware is strictly aimed at small-scale (<20 detector module) readouts.

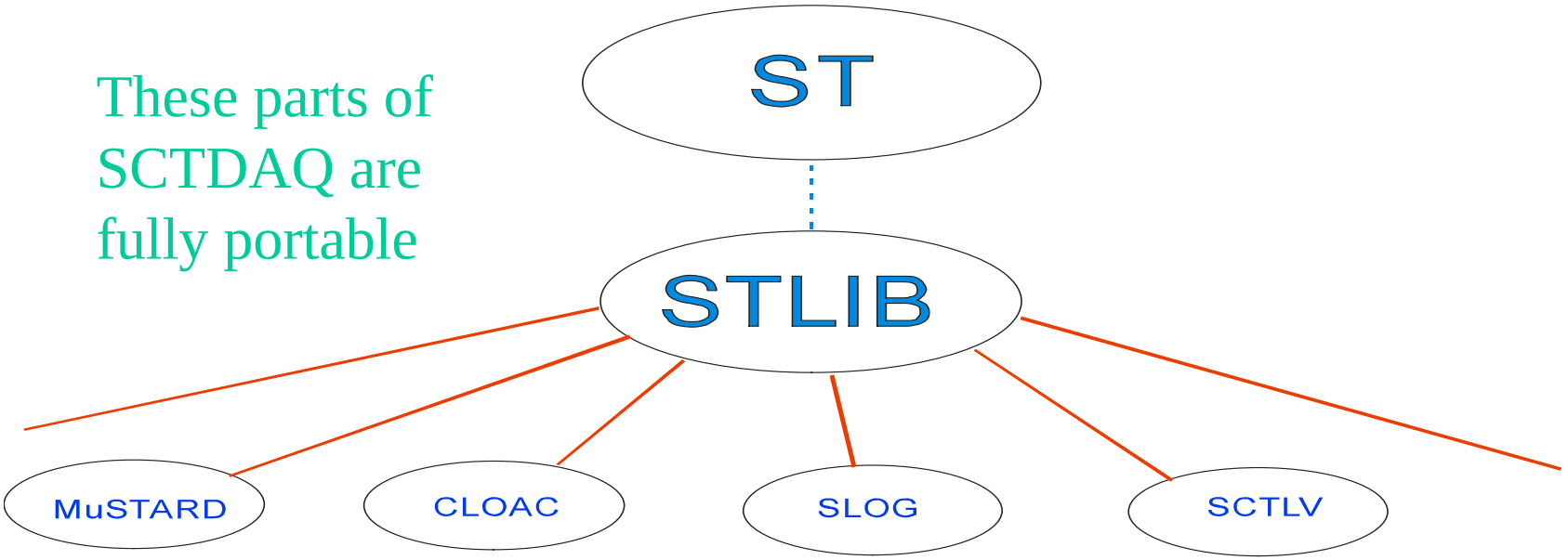
- SCTDAQ was developed within this hardware framework - i.e. the software was neither designed for, nor has been tested in, large-scale systems.

Example: CERN systemtest



Basic SCTDAQ structure (1)

These parts of SCTDAQ are fully portable



ST - simple command line interface for testing and environments without root

STLIB - ABCD chip handling functions
- map detector modules to VME modules using information from configuration file
- write results to ASCII N-tuple for “offline” analysis
- high level operations (e.g. set up, generate and readout triggers)

MuSTARD etc. - hardware libraries for VME modules

Basic SCTDAQ structure (2)

•All libraries in previous diagram are in ANSI-C

•Though main platforms used are Windows 95 and NT, EP/LX (Real Time UNIX) on RAID processor has been used.

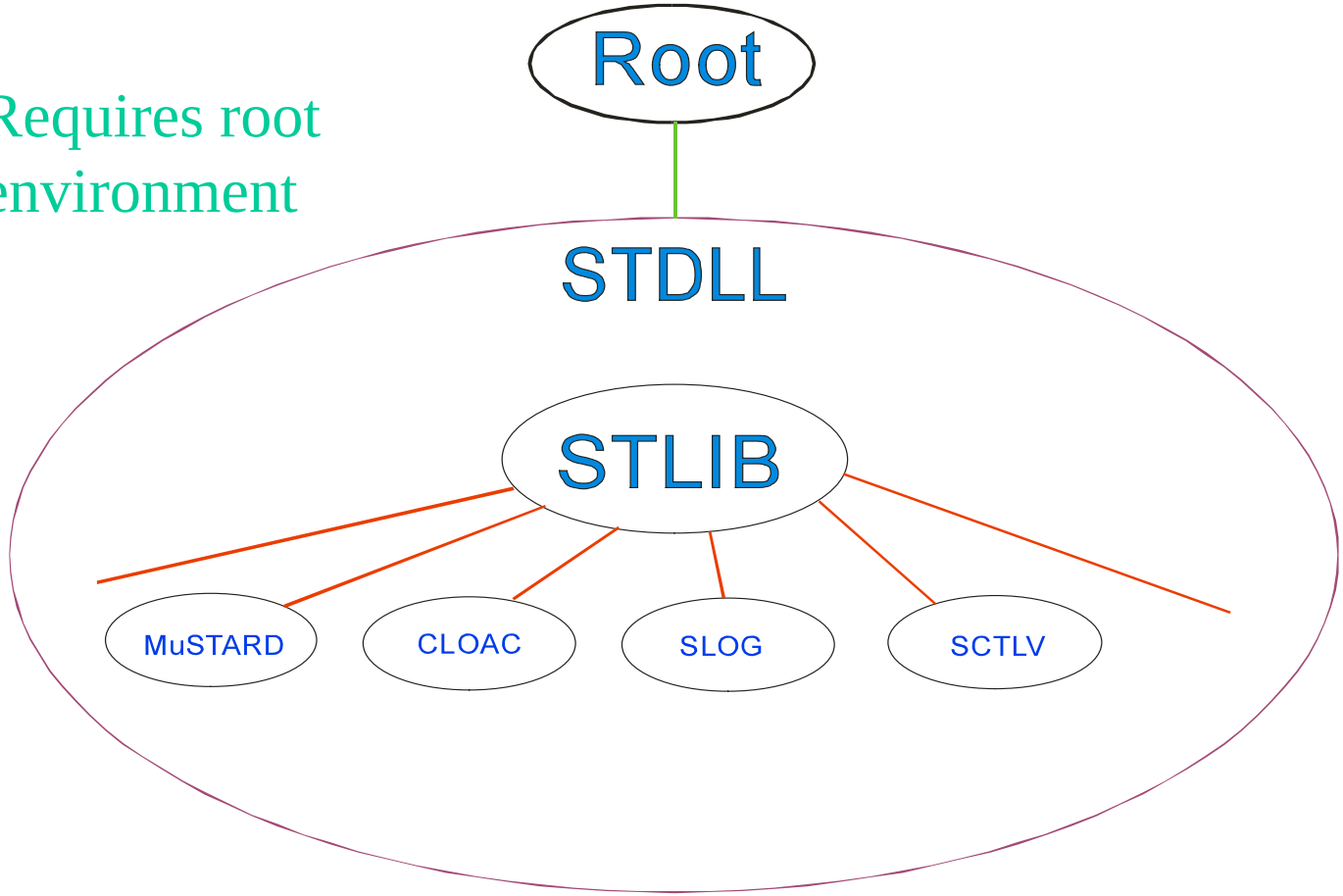
•Hardware libraries were also used in RD13 DAQ (DAQ -2??) in test beam work up to 1999.

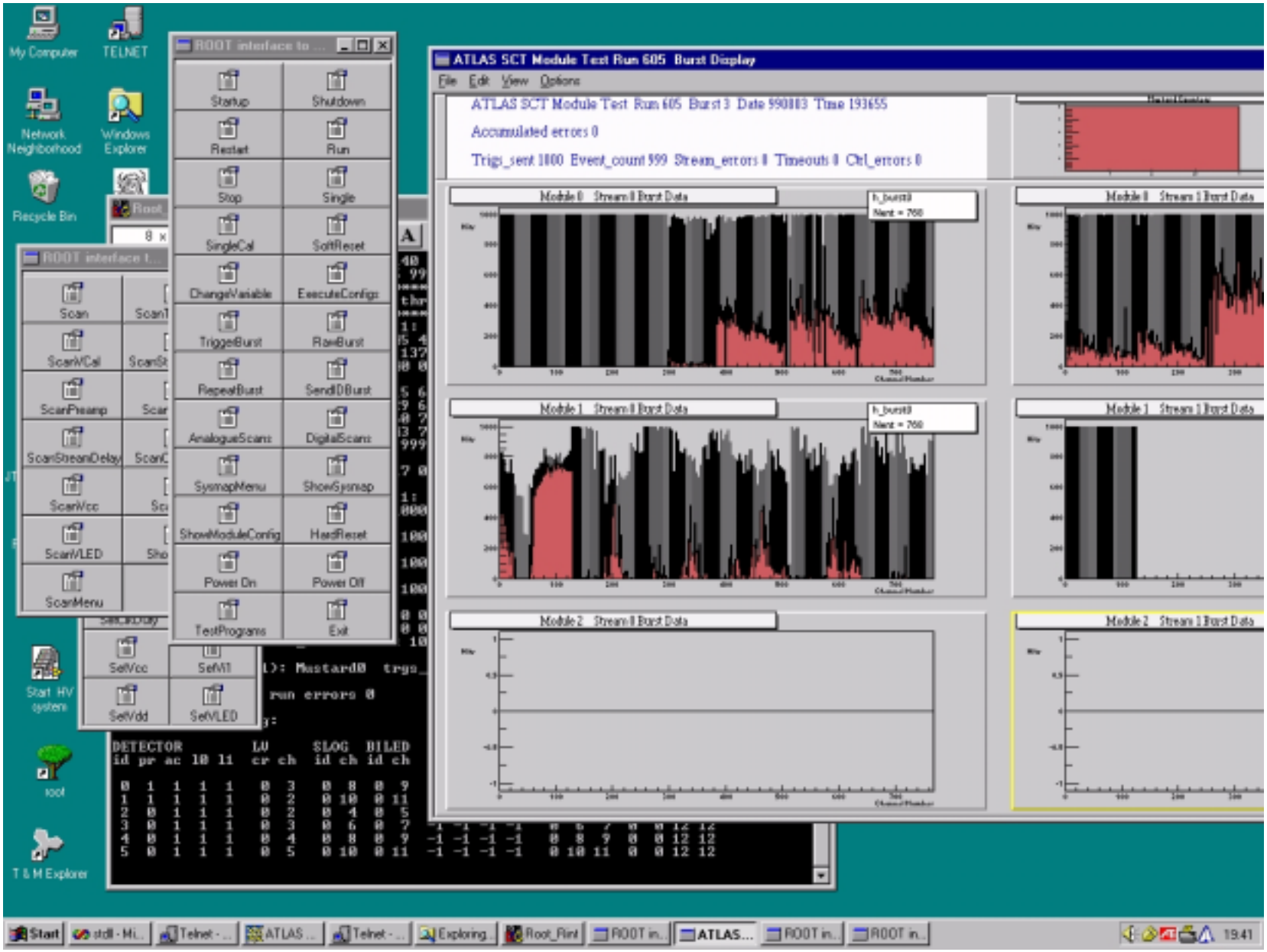
- •VME i/o in well-defined locations, and simple access only, so simple to move to alternative interface (as long as memory-mapped approach is used).

•Structure should allow insertion and replacement of new libraries in a clean way (e.g. already done with BiLED==> OpTIF). This should permit ROD and TIM software to be included.

Basic SCTDAQ structure (3)

Requires root environment




ATLAS SCT Module Test Run 605 Burst Display
File Edit View Options
ATLAS SCT Module Test Run 605 Burst 3 Date 990803 Time 193655
Accumulated errors 0
Trigs_sent 1000 Event_count 999 Stream_errors 0 Timeouts 0 Ctrl_errors 0
Module 0 Stream 0 Burst Data
Module 0 Stream 1 Burst Data
Module 1 Stream 0 Burst Data
Module 1 Stream 1 Burst Data
Module 2 Stream 0 Burst Data
Module 2 Stream 1 Burst Data
ROOT interface to ...
Startup
Shutdown
Restart
Run
Stop
Single
SingleCal
SoftReset
ChangeVariable
ExecuteConfigs
TriggerBurst
RawBurst
RepeatBurst
SendDBurst
AnalogueScans
DigitalScans
SysmapMenu
ShowSysmap
ShowModuleConfig
HardReset
Power On
Power Off
TestPrograms
Exit
Scan
ScanVCal
ScanPreamp
ScanStreamDelay
ScanVcc
ScanVLED
ScanMenu
SetVcc
SetVdd
SetVLED
DETECTOR
SLOG
DILED
My Computer
TELNET
Network Neighborhood
Windows Explorer
Recycle Bin
Start HV system
root
T & M Explorer
Start
Telnet - ...
ATLAS ...
Exploring ...
ROOT in ...
ATLAS...
19:41

Basic SCTDAQ structure (4)

ST.cpp - basic root user macro - loads stdll, optionally initialises the VME interface, defines menu buttons, and some of the code for the menu functions.

STDLL - includes all of stlib and hardware libraries, PLUS C++ code for:

- majority of menu functions
- histogram definitions and filling
- “bursts” of triggers
- curve fitting
- some CLOAC functionality
- other “higher level” functions

Organisation of code not ALWAYS 100% logical!

A few words on root

- Can be used to provide a complete DAQ environment (e.g. in MINOS)
- Advantages of using root with SCTDAQ:
  - GUI framework (menus) and histogram presentation comes “for free”
  - Portable across platforms supported by root - i.e. Windows, Linux, Unix. User C++ code for SCTDAQ is not machine dependent
  - Users can add extra functionality relatively easily by writing their own macros (they don’t need to delve into the full mysteries of the system).

Menus

- Large number of menu options!
- They have appeared in the light of experience with real modules
- Significant number of menu functions are designed for the handling of a few modules with idiosyncrasies.
- Also, SCTDAQ is aimed at characterising modules in a small lab. environment, not data acquisition, and hence does not have (for example) sophisticated error recovery.

- However:
  - the “TB extensions” (see a little later in the talk) provide a simple DAQ within the SCTDAQ framework, and were successfully used in the 2000 test beam

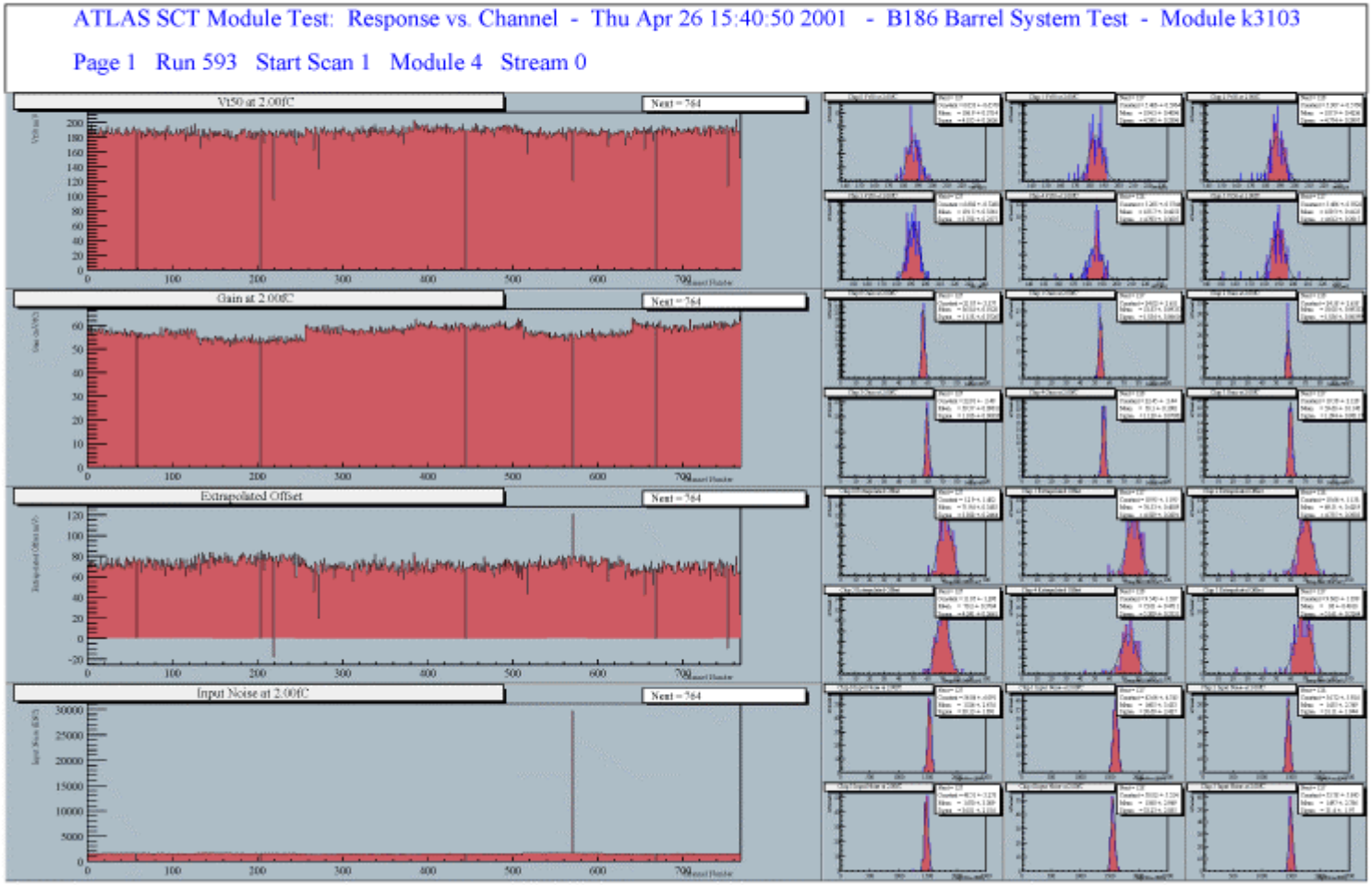
Main Menu

- Startup
- Shutdown
- Restart
- CloacMenu
- Run
- Stop
- ChangeVariable
- ChangeTrigger
- ExecuteConfigs
- TriggerBurst
- TriggerBurst2
- RawBurst
- DecodedBurst
- VTriggerBurst
- DecodedVBurst
- DumpVBurst
- DumpBurst
- SendIDBurst
- RepeatingBurst
- ABCD Tests
- SimpleScans
- KwikPlot
- ShowSCurves
- ShowCounters
- SysmapMenu
- ShowSysmap
- ShowModuleConfig
- ShowModuleTrims
- ShowModuleRC
- HardReset
- OPTO TX On
- OPTO TX Off
- OPTO TX Status
- HV RampUp
- HV RampDown
- HV Status
- LV On
- LV Off
- LV Status
- Recover Trips
- DCSQuery
- DCS->Log
- TestPrograms
- Status
- Documentation
- Exit

Subsidiary Menus

- CloacMenu
  - interfaces directly to CLOAC functionality
- ChangeVariable
  - set chip parameters, delays on inputs and outputs, etc.
- ChangeTrigger
  - alter trigger details
- ABCD Tests
  - e.g. check the redundancy features, automatic TrimDAC trimming
- SimpleScans
  - simple 1D scans (e.g. threshold scan)
- TestPrograms
  - diagnostic programs for VME readout modules
- SysmapMenu
  - command line interface to sysmap menu

Scan example - response curve



Test Beam Extensions(1)

- Used in test beam from 2000 (replacing RD13 DAQ and RAID)
- Mainly work of Gareth Moorhead
- Contructed as an “add-on” to previously-described SCTDAQ, so relies on that to set up detector and standard VME modules:
  - Deal with extra hardware in test beam:
    - Beam telescope (analogue readout, using DSP-based Siroccos (IRAMs))
    - CAEN V488 TDC (trigger timing relative to 40MHz clock)
    - CAEN V262 I/O unit (software polling for beam triggers)
  - Structure similar to STDLL/STLIB - code built into TBDLL, which includes:
    - C++ code for beam telescope, data monitoring, run handling, general bookkeeping
    - TBLIB: C code specific to test beam (special hardware, event read out and writing)
    - Libraries for test beam hardware
    - STLIB and hardware libraries from SCTDAQ

Test Beam Extensions(2)

- Invoked within root framework after standard SCTDAQ startup has been done (via ST.cpp)
- Macro TB.cpp loaded - this contains
  - TBStart - loads TBDLL, initialises the environment, creates the extra histograms required for test beam, and sets up the hardware appropriately for test beam
  - TBRun - starts a data-taking run
  - TBScan - starts a set of runs to scan a range of values for a specific variable - very useful for unattended running (or at least running with minimal intervention).
- Command line interface for TB commands.

- Since existing SCTDAQ menus are available, operation is consistent with lab use - no new way of working to learn, and all “tricks” to persuade detector modules to behave, are available automatically.

Use with ROD/BOC/TIM?

- Reasonably clean mapping from existing to new hardware:
  - CLOAC+SLOG(+OpTIF) ==> TIM+Outgoing (control) connections from ROD/BOC
  - MuSTARD(+OpTIF) ==> Incoming (data) connections to ROD/BOC

- TIM functionality is a superset of CLOAC's (to a good approximation)
- BOC functionality is a superset of OpTIF's (...)
- SLOG functions (slow commands to front end ASICs):
  - in short term will be replaced by sending a bitstream via a ROD primitive
  - in longer term, we will probably have bitstreams predefined in the DSP and just issue the download instruction
- MuSTARD functions (reading data, optional decoding and histogramming):
  - already implemented or planned for ROD

Use with ROD/BOC/TIM?

- Hence, for short term, seems a good solution for a ROD DAQ
  - exists and has been tested by many users
  - familiar interface for users
  - sections (e.g. chip-handling code, analysis macros), which are based on real experience with detector modules, would probably be reused anyway
- However, not a long term solution
  - not designed for 100's of modules (reaching the practical limits in places already??)
  - intended for the characterisation and control of a handful of prototype detector modules, so lots of scans, module resets, trip recovery etc. in menus, but not much emphasis on long term monitoring of a large number of modules, nor of automated recovery of modules. (TB does attempt this, but not altogether successfully).
  - needs significant rewriting for a final system - best to start from scratch?

Conclusions

- SCTDAQ seems a reasonable starting point for an interim ROD DAQ, but also needs the Test Stand software to control the ROD.
- There are clear areas where parts of SCTDAQ could be reused in the longer term:
  - chip-handling code
  - analysis macros
- Ability to add user scripts within root framework is attractive and something similar should be provided for the final design.
- One caveat - initial systemtest work will involve debugging of readout hardware, not dealing with real detector modules, and SCTDAQ is probably not suitable for this (use Test Stand software and ad-hoc programs).