CDF/DOC/SEC_VTX/PUBLIC/6163
November 1, 2002
Version 1.0

Silicon Monitoring with SVXMon

Igor Volobouev, Henri Bachacou
Lawrence Berkeley National Laboratory

Abstract

SVXMon is a monitoring program used for online diagnostics of the CDF silicon tracker in Run II. Its main purpose is to verify data integrity, accumulate various statistics, diagnose problems, and present a coherent set of error reports and detector/DAQ performance plots to the silicon experts and the shift crew.

Contents

1 Introduction

CDF monitors its data quality in real time using the so-called *Consumer Framework* [1]. In this framework, a fraction of events is made available for immediate analysis to a collection of programs known as "consumers" [2]. These programs run on various nodes in the CDF online computing cluster and process events which are continuously served over the network. The results, usually in the form of histograms, are sent to one or more online display programs run by experts or by the shift crew.

SVXMon is a consumer which monitors the quality of the silicon tracker data. It keeps track of all problems found by the silicon bank unpacker and performs a variety of data integrity checks. For each silicon strip SVXMon accumulates the number of hits and the first four moments of the pulse height distribution. These statistics are used to create plots of occupancies, average pulse heights, distribution shapes, etc. The results can be viewed with any degree of detail desired, from layers and barrels down to silicon ladders, chips, and individual strips. The program accumulates statistics both since the beginning of the run and in a recent time window. SVXMon's built-in data analysis capabilities can be used to diagnose various silicon tracker and/or DAQ problems online and to detect adverse changes in the running conditions. Offline analysis of SVXMon histograms and error reports allows for efficient tracking of silicon system problems from run to run.

The subsequent sections of this note are intended to serve both as a general program description and as a detailed reference manual. Feel free to browse them at whatever level of details you are comfortable with.

2 Data Validation

Rather than checking the contents of the raw silicon banks directly, SVXMon works with container objects created by the silicon bank unpacker. The unpacker generates a diagnostic status word for every silicon half-ladder (HDI) included in the run, and SVXMon inspects these status words and counts the number of errors seen. The error statistics are maintained per HDI and error type. From time to time SVXMon generates corresponding error messages and sends them to the error logging facility described in section 3. The frequency of error messages of the same type generated for a given detector is dynamically prescaled in order to reduce the degree of message redundancy but still remind the users about the problems. At the end of each run, a summary message is generated for every HDI and error type with the total error count. The errors detected by the silicon unpacker and the corresponding messages produced by SVXMon are listed in Appendix D together will all other types of SVXMon messages.

In a typical online monitoring configuration, SVXMon also performs a number of other checks of silicon data integrity in every event:

- Identifies readout chips whose pipeline has lost synchronization with the rest of the detector.
- Finds invalid ADC values present in the data stream due to chip malfunctions, optical transmission errors, or problems with pedestal subtraction in the FIBs.
- Monitors error words generated by VRB hardware.
- Checks the consistency of several DAQ-related quantities between SVX and L00/ISL

(SIXD and ISLD banks).

- Verifies that all detectors included in the run have been read out by the DAQ (and that detectors not included are not read out).

In addition to the deterministic checks listed above, SVXMon can perform an analysis of pulse height distributions and identify misconfigured or malfunctioning readout chips, problems with ladder bias voltage, *etc.* This analysis, however, depends on a set of cuts which requires a substantial amount of tuning. This topic is covered in more detail in section 5.

SVXMon can be configured to dump events with problems to disk for subsequent analysis by experts. The number of saved events is limited by SVXMonModule parameters "MaxPassPerRun" and "MaxPassTotal", and can be fine-tuned for specific problem types using the "SIXDMon dumplimit" command described in section 7.2. The events are writen out by one of the standard AC++ output modules, while SVXMonModule works as a filter. Unfortunately, the output modules are not run-oriented, so it is impossible at this time to ensure that their buffers are flushed to disk at the end of each run. The events are guaranteed to appear on disk only when the SVXMon job is finished.

3 Error Reporting

SVXMon messages are received, reported, and stored by a Tcl/Tk [3] script called "SVX Error Logger". This script provides a convenient and intuitive GUI for collection and analysis of error messages produced by SVXMon and, potentially, by other CDF online monitoring programs. SVX Error Logger receives messages over a TCP/IP link and displays them to the user in a window reminiscent of the Netscape mail reader. The logger incorporates facilities for message viewing, searching, sorting, saving, restoring, distributing, and for generating run summary statistics. The SVX Error Logger program is compatible with the ZOOM Error Logger package [4] via an adapter class called SvxErrorLogger which sets up all necessary communications and performs the message transfers. The logger GUI is implemented with Tcl/Tk version 8.3 (or newer). It also needs Tcl/Tk extensions BLT [5] and Signal [6]. The required Tcl/Tk libraries, extensions, and executables are installed both on fcdfsgi2 and on B0 online machines, but they are not part of the official CDF software distribution.

3.1 SVX Error Logger GUI

The SVX Error Logger runs in a separate window which can be displayed locally or over the network using X Window System. The components of the logger GUI window are illustrated in Fig. 1. The detector selection frame shows a collapsible hierarchical tree of detector systems and components for which at least one error message has been received. A small icon near the detector name shows the highest error severity for all messages received for this particular detector and all its subdetectors. The severity levels are the same as in the ZOOM Error Logger. Click on the "Severity Legend" item in the "Help" menu to see which icon corresponds to which severity level. The naming of detector components mimics the UNIX directory structure. Clicking on a detector system causes all error messages for that system to be displayed in the message selection frame. By default, only the messages for

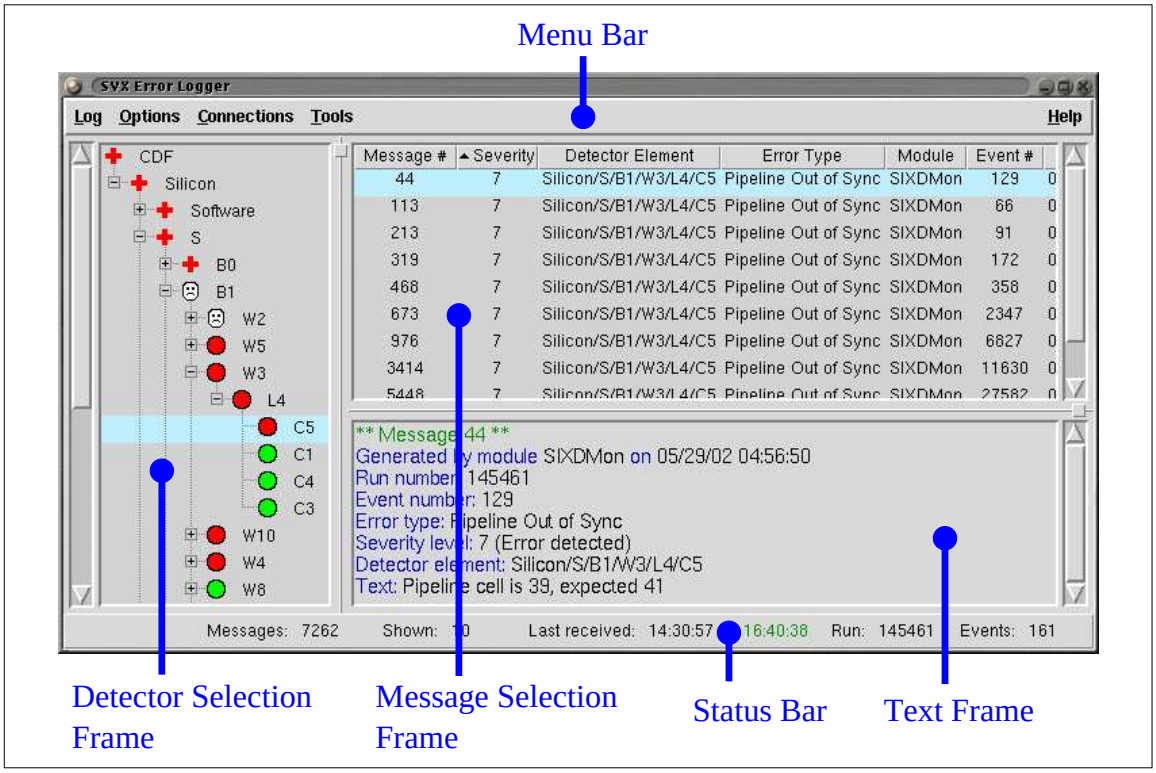


Figure 1: SVX Error Logger GUI

this particular system are displayed but not for its subsystems. To include all subsystems, turn on the "Recurse Subdetectors" option in the "Options" menu.

Messages displayed in the message selection frame can be quickly sorted by number, severity, error type, *etc.* Just click on the corresponding column title. Subsequent clicks on the same title will change the ordering from ascending to descending and back. If some columns are obscured, you can make them visible by dragging the message selection frame from side to side with the middle mouse button or by increasing the size of the whole GUI window. Clicking on the message itself will cause the complete message info to appear in the text frame. You can click on the frame with your right mouse button and then use up and down arrows on your keyboard to move from one message to another.

Instead of looking at the error messages for some detector system, one can observe the latest error messages as they arrive by selecting the "Display Last Message" option from the "Options" menu. In this case the text part of the last message is displayed automatically in the text frame.

Entries in the "Options" menu called "Unique Message Types" and "Unique Messages" allow the user to suppress messages which are similar to other messages already displayed in the message selection frame. If "Unique Message Types" option is selected then only those messages are shown which differ from every other message in at least one of the following attributes: severity, detector element, error type, or module. In case of the "Unique Messages" option the list of compared attributes also includes the message text. Naturally, only one of these options can be selected at a time.

Basic file saving and loading operations are performed from the "Log" menu. This menu is also used to load the list of ladders with known problems, delete all messages, or exit the program.

The status bar at the bottom of the window displays the following (from left to right):

- Current program activity (or an empty string if the program is not doing anything).
- Total number of error messages received.
- The number of error messages shown in the message selection frame. The background color for this number may turn yellow. This means that the set of messages shown in the frame is no longer current. Click on the yellow field to update the frame.
- The time when the last message arrived over the network. On start-up, this time is initialized to the program start time.
- Current time (in green).
- CDF run number.
- Number of events processed by SVXMon in the current run.

3.2 Message Searching

To search for specific error messages, you need to click on the "Find Messages" item in the "Tools" menu. This will bring up the search interface window shown in Fig. 2. This

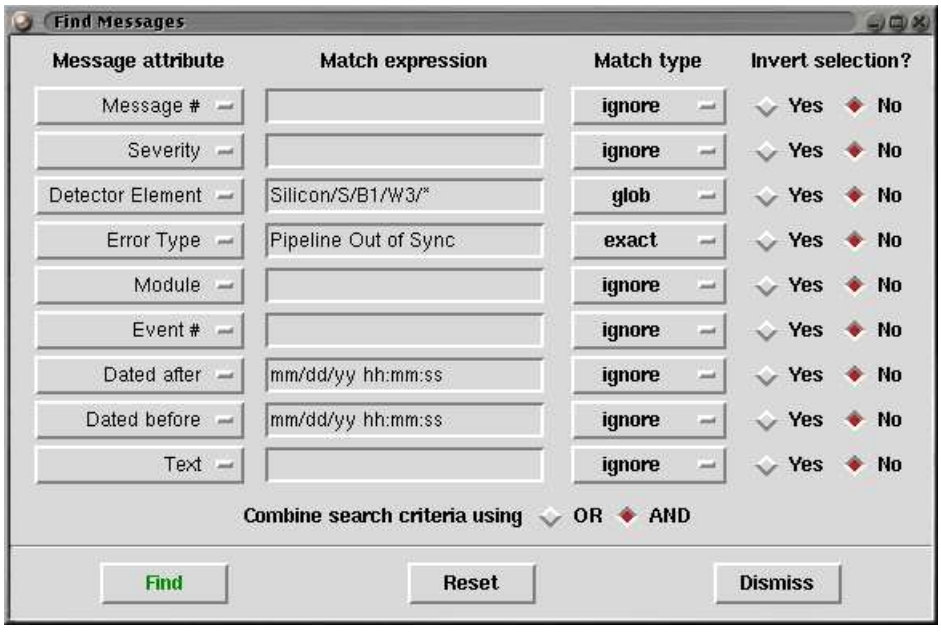


Figure 2: SVX Error Logger Message Searching Interface

interface allows you to select messages using different message fields and search types. The "Match type" column determines which fields to use and how to establish the match with the entries in the "Match expression" column. The following match types are supported:

ignore — Means that this message field is not used in the search.
exact — The content of the message field must exactly match the expression provided.

range — Used only with fields which contain integers (*e.g.*, Severity). The match expression in this case must consist of two integers separated by a space, in non-decreasing order. Both upper and lower limits will satisfy the condition, as well as all integers in between.

select — Used only with dates. Opposite of **ignore**.

substring — Used only with message fields which contain strings (*e.g.*, Error Type). The message will satisfy the search condition if the match expression is a substring of the relevant field.

glob — String fields only. The program will perform shell-like globbing of the relevant message field using the tcl "string match" command.

regexp — String fields only. The program will search the relevant message field using the tcl regular expressions syntax which is similar to that used by the UNIX "grep" utility.

All match expressions used with string fields are case sensitive. It is possible to use the same message field more than once in the same search — every row in the "Message attribute" column is actually a menu which allows you to choose any field in any row. The "Invert selection?" column allows you to suppress messages which satisfy some search criteria instead of selecting them. When you click the "Find" button, messages which satisfy all your search criteria will be displayed in the message selection frame of the main window. You can customize your search even further by selecting "Unique Message Types" or "Unique Messages" item from the "Options" menu.

The most recent search criteria will also be applied to the incoming messages until you click on one of the systems in the detector selection frame or select the "Display Last Message" option.

3.3 Summary Statistics

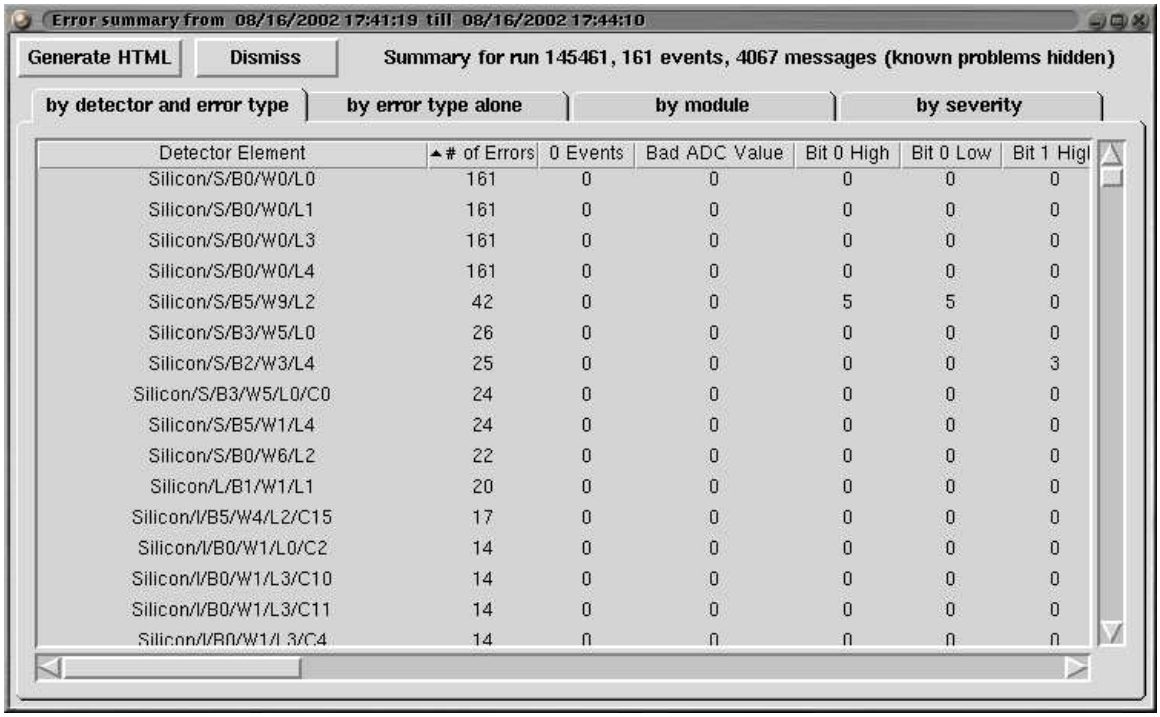
The summary statistics for the accumulated error messages can be viewed by clicking on the "Statistics" entry in the "Tools" menu. The program generates several tables of error counts and pops up a new window which displays these tables. An example window is shown in Fig 3. Each table takes one page in a notebook-like interface. To bring a table forward click on its tab. As in the message selection frame of the main GUI window, the tables may be sorted using any column by clicking on the desired column title.

Error messages about ladders with known problems may be suppressed in the summary by loading a list of bad ladders. Use the "Known Pb's" entry in the "Log" menu to load the list and turn on the "Hide Known Problems" option in the "Options" menu before generating the summary. The file with the list of bad ladders is just a text file containing one FIB id per line (in hex). Empty lines are ignored. Here is an example of such a list:

```
e500
e501

f140
```

During normal online operation, the list of known problems is updated automatically when SVXMon starts, based on the contents of the Silicon Problem Database [7]. This list is used



Detector Element	# of Errors	0 Events	Bad ADC Value	Bit 0 High	Bit 0 Low	Bit 1 Hig
Silicon/S/B0/W0/L0	161	0	0	0	0	0
Silicon/S/B0/W0/L1	161	0	0	0	0	0
Silicon/S/B0/W0/L3	161	0	0	0	0	0
Silicon/S/B0/W0/L4	161	0	0	0	0	0
Silicon/S/B5/W9/L2	42	0	0	5	5	0
Silicon/S/B3/W5/L0	26	0	0	0	0	0
Silicon/S/B2/W3/L4	25	0	0	0	0	3
Silicon/S/B3/W5/L0/C0	24	0	0	0	0	0
Silicon/S/B5/W1/L4	24	0	0	0	0	0
Silicon/S/B0/W6/L2	22	0	0	0	0	0
Silicon/L/B1/W1/L1	20	0	0	0	0	0
Silicon/I/B5/W4/L2/C15	17	0	0	0	0	0
Silicon/I/B0/W1/L0/C2	14	0	0	0	0	0
Silicon/I/B0/W1/L3/C10	14	0	0	0	0	0
Silicon/I/B0/W1/L3/C11	14	0	0	0	0	0
Silicon/I/B0/W1/L3/C4	14	0	0	0	0	0

Figure 3: SVX Error Logger Message Summary (offline example)

by SVXMon for masking chips with known problems on the pipeline status map (section 4.8) and the chip status map (section 5).

The "Generate HTML" button may be used to write the contents of all tables into an HTML file. A pop-up dialog window will allow you to choose the file name. This file can be later viewed with a web browser.

Click on the "Dismiss" button to close the window. The same effect can be achieved by using the window manager "delete" action (usually by clicking with the left mouse button on a small diagonal cross in the upper right corner of the window).

A different type of summary can be generated by clicking on the "Run Summary" button in the "Tools" menu. This summary is a simple text file which can be later printed on a printer or viewed in a terminal. A pop-up dialog window will allow you to choose the file name. A special file name "console" may be used to display the summary information in the text frame instead of writing it to disk. In this summary, the error messages are sorted by the detector element, for example:

```
SB3W7L4
        Bit Error (not 0/1) | 1 | This error was present in 1 events out of 156
        Unknown Channel | 1 | This error was present in 1 events out of 156
        ff or 7f Error | 2 | This error was present in 2 events out of 156
```

After the detector element, all error types encountered for this element are listed together with the number of errors seen (separated by vertical bars) and the text of the last message of this type. For SVXMon, this last message is usually (but not always) a summary message generated at the end of a run.

Yet another summary can be generated by clicking on the "JPF Run Summary" button in the "Tools" menu (this summary format was suggested by Juan Pablo Fernandez). In this summary the error messages are sorted by type and written to a file or the text frame like this:

```
(*)     SB0W4L3C6(3),SB0W4L3C7(2),SB0W4L3C8(1),SB1W5L4C0(1),
        SB1W5L4C5(6),SB1W5L4C6(6),SB1W5L4C9(1),SB2W0L4C4(68),
        SB4W10L3C1(2), SB4W10L3C2(1),SB4W10L3C4(2)

        Invalid Cell Id
```

The number of errors of the given type is given in parentheses after each detector element.

When SVX Error Logger works online in tandem with SVXMon, it automatically generates three text files at the end of each CDF run. The files are named like this:

```
svxmon_RRRR_hhhh_PPPP.errlog
svxmon_RRRR_hhhh_PPPP.errsum
svxmon_RRRR_hhhh_PPPP.errtypes
```

where "RRRR" stands for the run number, "hhhh" for the host name of the machine on which the logger runs (with domain name stripped off), and "PPPP" for the process number of the logger. This naming convention ensures that several error loggers running simultaneously will not use the same file name. The file with extension ".errlog" contains the full list of error messages, the ".errsum" file contains the error summary by detector elements, and the ".errtypes" file stores the error summary by error type.

At the time of this writing, only the summary statistics GUI and the HTML summary support suppression of known problems by detector element. The text summaries always use the full set of messages.

3.4 SVX Error Logger Communications

Two or more SVX Error Logger programs can communicate to each other over the network. Loggers can be used both as "sources" and as "sinks" of messages. Each connection between two error loggers is used to pass messages in one direction only, from source to sink. The messages are, in fact, tcl commands processed by a "safe interpreter"[1] on the sink side. After a command is executed, the status ("ok" or "error") is sent back to the source. To keep networking simple and robust, the limits on the number of simultaneous connections have been set as follows: a logger can not have more than one source and four sinks connected to it. Although somewhat arbitrary, this restriction ensures that the messages can't go around in loops and that all logger programs remain reasonably responsive and don't get overburdened with message passing.

A connection between two error loggers can be established using the "Connections" menu. Either source or sink side can initiate the connection (act as a client) by clicking on the "Connect as Source" or "Connect as Sink" menu item. The side which is being

[1] "Safe interpreters" allow for safe execution of untrusted tcl scripts and for providing mediated access by such scripts to potentially dangerous functionality. Such interpreters ensure that untrusted scripts cannot harm the hosting application or disclose information stored on the hosting computer/application to any party. Please see Ref. [3], pp. 273 – 291 for more details.

connected to (the server) must have its "Accept Connections" option enabled. To establish a connection, you need to know the host name and the port number of the server. The port number can be provided on the command line when the SVX error logger starts. For example,

```
$CDFSOFT2_DIR/SvxDaqMods/test/ErrorLogger/start_logger 9234 &
```

tells the logger to accept connections on port 9234. If the port number is not provided on the command line, the default value of 9110 will be used. When the logger is started by SVXMon, the port number is determined by the SVXMonModule configuration parameter "ErrorLoggerPort". When the logger is running, its server port number and the number of active connections can be checked at any time by clicking on the "Show Connections" item in the "Connections" menu. When a connection is established, the source logger passes all messages accumulated by it at the time of the connection and all future messages to the sink logger, so that the set of messages presented to the users of both source and sink are identical.

The default SVX Error Logger behavior when its source of messages disconnects is to save accumulated messages into a file and exit. This feature ensures that the logger processes which are not doing anything are gracefully terminated. However, this may be inconvenient if you still want to view and analyze the messages. To disable automatic exiting, unselect the "Exit when Source Disconnects" option in the "Connections" menu.

Another error logger GUI connected to the current logger as a sink can be started on the same display using the "Duplicate Logger" entry in the "Tools" menu. Duplicate loggers are useful if you need to look at the results of more than one message search at a time. By default, duplicate loggers do not write a log file when they exit.

The "Secure Parser" option in the "Connections" menu should normally be enabled. When it is not enabled, it becomes possible to connect to the server and execute tcl commands inside the top-level interpreter which runs the program. This is very useful for debugging but may leave the program vulnerable to unauthorized access.

4 SVXMon Plots

SVXMon can create four types of plots: long-term, short-term, history, and periodic. Long-term plots display various silicon tracker statistics accumulated since the beginning of the run. Short-term plots display statistics accumulated during the last few minutes of data taking or over a few most recent events. These short-term plots are often called "snapshot plots" in the rest of this document. History plots can show the time history of any quantity for which there is either a long-term or a short-term plot. Periodic plots display sequences of various silicon quantities averaged over a fixed number of events. These plots are similar to history plots in purpose but occupy less memory, and creation of the relevant ROOT histograms is delayed until the end of a job.

Each SVXMon plot is usually placed on its own canvas which provides some useful auxiliary information such as the CDF run number, the number of events accumulated so far by SVXMon, plot update time, *etc.*

Due to the large amount of information passing involved, SVXMon plots are not updated each event. Instead, the updates are performed according to a simple scheduling algorithm.

This algorithm is described in section 7.3 together with the meaning of SVXMon parameters "neventsHardUpdate", "neventsSoftUpdate", "timeHardUpdate", and "timeSoftUpdate". The snapshot intervals are also set with a similar algorithm, only parameter names are different ("snapEventsHardUpdate", *etc*). However, it is sometimes useful to instruct SVXMon to update its plots and to make a snapshot every event. In this mode of operation one can display silicon data in real time, event by event, as soon as they are read out by the DAQ.

SVXMon plots may be booked either at the beginning of a job in its configuration file or in the middle of a run using an interactive prompt (section 7 provides more details about SVXMon configuration mechanisms). All plots and histograms are created with a special command named "histo" added to the tcl interpreter by SVXMon. The rest of this section gives a detailed description of the "histo" command and the objects it creates.

4.1 Strip Plots

SVXMon can create several types of plots in which various silicon-related quantities are monitored by strip. One-dimensional plots of some quantity vs. the strip number may be created by the following command:

histo strips *detector histoTypes makeSnapshots*

Here, *detector* is a five-element tcl list which specifies one face of a silicon half-ladder [8]. *histoTypes* specifies the quantities to plot. It must be a tcl list which contains one or more of the following keywords:

"occupancy" — Plots strip occupancy in percent vs. the strip number.
"vrbocc" — Plots VRB (level 1) strip occupancy in percent (the CDF silicon DAQ may require a configuration adjustment to provide access to this information).
"nevents" — Plots the number of events in which a strip has registered a hit. This plot differs from the occupancy plot only by normalization.
"mean" — Plots average strip pulse height in ADC counts.
"stdev" — Width of the pulse height distribution in ADC counts.
"skewness" — Pulse height distribution skewness.
"kurtosis" — Pulse height distribution kurtosis.
"bad" — Plots 1 for a "bad" strip, 0 for a "good" strip.
"discarded" — Plots 1 for a discarded strip and 0 for a strip which doesn't have the "discard" tag. See the description of "svx badstrips" command in section 7.4 for a discussion of bad and discarded strips.
"newbad" — Plots 1 for strips which have the "bad" tag but do not have the "discard" tag. This plot and the next one may become useful in the future if a real-time bad strip diagnostics is implemented in SVXMon.
"newgood" — Plots 1 for strips which have the "discard" tag but do not have the "bad" tag.
"dnoise" — Plots dnoise in ADC counts. Dnoise is the standard deviation of the pulse height difference between adjacent channels divided by $\sqrt{2}$. This plot makes sense only when the half-ladder is used in "read all" mode.

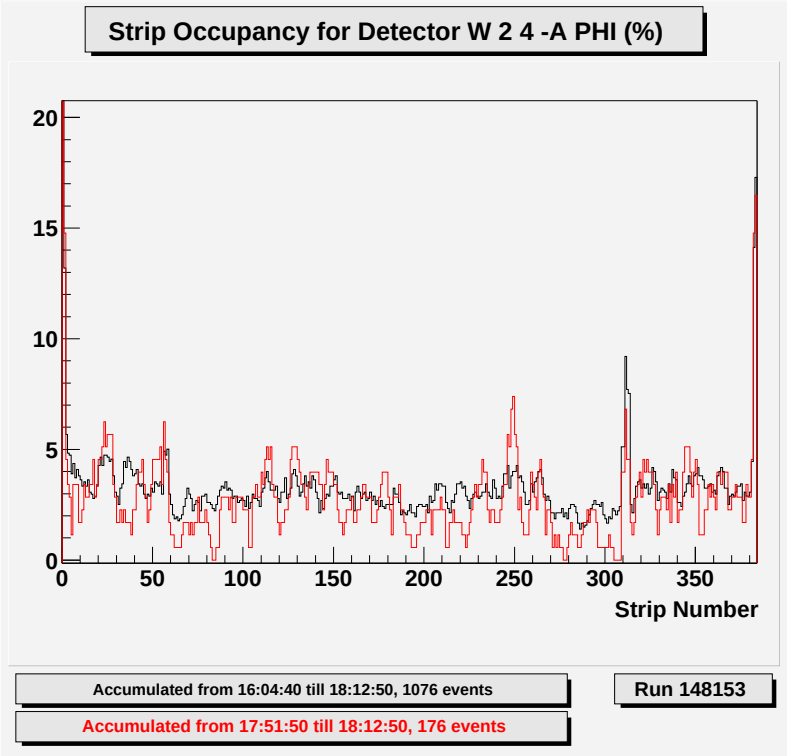


Figure 4: Example plot of occupancy vs. strip number for the phi side of the SVX half-ladder in the west barrel, west half-barrel, layer 2, wedge 4. Together with the plots shown in Fig. 5, this plot has been booked with the command "histo strips {west west 4 2 phi} {occupancy mean stdev} 1". Short-term plot (red) is drawn on top of the long-term plot (black).

The command will create as many plots as the number of elements in the *histoTypes* list. *makeSnapshots* is a boolean argument which specifies if a short-term plot should be overlayed on top of the long-term one. This argument should be in agreement with the SIXDMon parameters "snapshotStrategy" and/or "vrbOccupancySnapshots" (described in section 7.3). If "makeSnapshots" is "true" while "snapshotStrategy" is set to 0 then the short-term plots will be created but they will never be filled — obviously, this is not a useful configuration. Examples of plots produced by the **histo strips** command are shown in Fig. 4 and 5.

In a typical online monitoring configuration, SVXMon creates plots of strip occupancies, mean charge, and RMS charge for every half-ladder side in SVX/ISL and every z segment in L00. Of course, neither experts nor consumer operators are expected to view about 4000 such plots. Rather, the strip data are processed with SVXMon online (section 5) and offline (section 6) quality control algorithms, and the strip-level information for some ladders may be reviewed when problems are detected.

SVXMon can also create 2-d plots of certain quantities vs. the strip number on the half-ladder side and another coordinate defined by the value of a certain DAQ-related parameter. These plots are booked with the following command:

histo strips2d *paramName detector histoTypes snapshotStrategy*

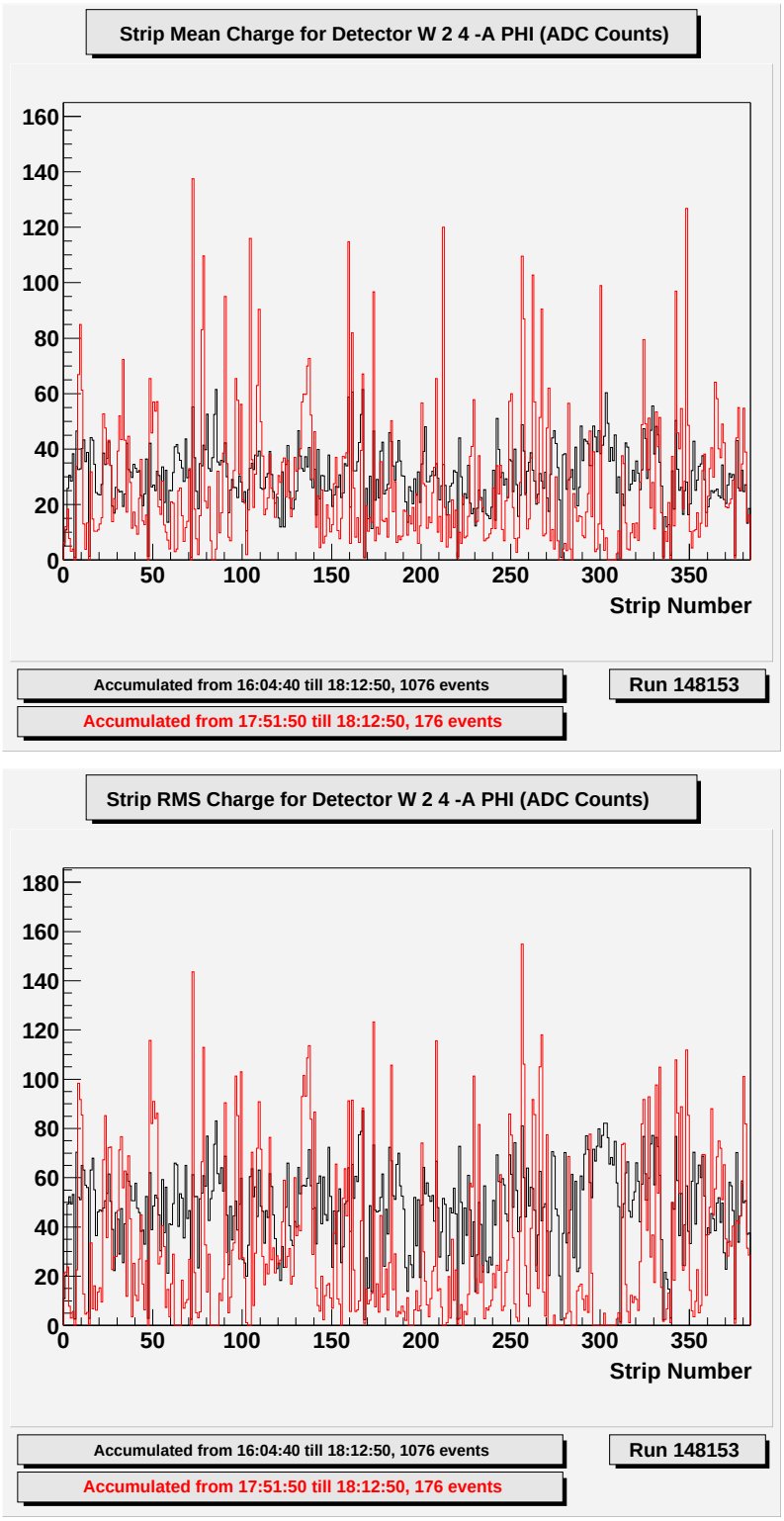


Figure 5: Example SVXMon plots of average and RMS readout charge vs. strip number.

paramName determines the quantity to use for the y histogram axis (the strip number will be used for the x axis). Valid *paramName* values are:

"adc" — used to produce pulse height distributions
"cellid" — pipeline cell id
"bemode" — back end mode of the SVX readout chip
"bunchx" — SRC bunch crossing number
"tl1a" — time since the last L1 accept

detector is the standard SVXMon half-ladder side specifier [8]. *histoTypes* specifies the list of quantities to histogram. The following keywords may be used in this list: "occupancy", "nevents", "mean", "stdev", "skewness", "kurtosis", and "dnoise". The meaning of the keywords in the same as in the **histo strips** command. Note that specifying "dnoise" as one of the quantities only makes sense when the detector is used in "read all" mode. *snapshotStrategy* is an integer which defines how many snapshots of the 2-d dataset will be used. Set it to 0 if you don't want to create any short-term plots, to 1 if you want one snapshot, and to 2 if you want to interleave two snapshots on a short-term plot. The snapshot updating schedule is specified for all histograms simultaneously using monitor parameters "snapTimeSoftUpdate", "snapTimeHardUpdate", "snapEventsSoftUpdate", and "snapEventsHardUpdate" (section 7.3). Snapshots of the 2-d plots are *independent* from the global "snapshotStrategy" parameter of the monitor.

The command will create as many canvases as the number of elements in the *histoTypes* list. Each canvas will have one or two histograms, depending on the value of the *snapshotStrategy* parameter. An example pulse height distribution created with the **histo strips2d** command is shown in Fig. 6, and a plot of RMS charge for each pipeline cell of a readout chip is shown in Fig. 7.

The **histo strips2d** command returns a histogram handle command [9]. The only useful thing you can do with the returned handle is to suppress all histogram entries for one of the parameter values. This feature is useful for displaying data from so-called "deadtimeless scans" when two L1 accepts are sent to the silicon front-end at a set of predefined intervals in order to study the effect of DAQ signals on the pedestal and noise. Here is an example of the handle usage:

```
set detector {west west 4 2 phi}
set histoTypes [list nevents mean stdev dnoise]
set handle [histo strips2d bunchx $detector $histoTypes 0]
$handle configure skipParamValue 5
```

4.2 Occupancy, Mean Charge, *etc.* vs. DAQ Parameters

If, for a given half-ladder, you are only interested in the dependence of some silicon quantity (such as occupancy or mean collected charge) on some DAQ parameter and not on the strip number, you can use the **histo daqparam** command:

histo daqparam *paramName detector histoTypes snapshotStrategy*

Histograms booked by this command are essentially sums or averages over all strips of corresponding **histo strips2d** type histograms described in the previous subsection. The

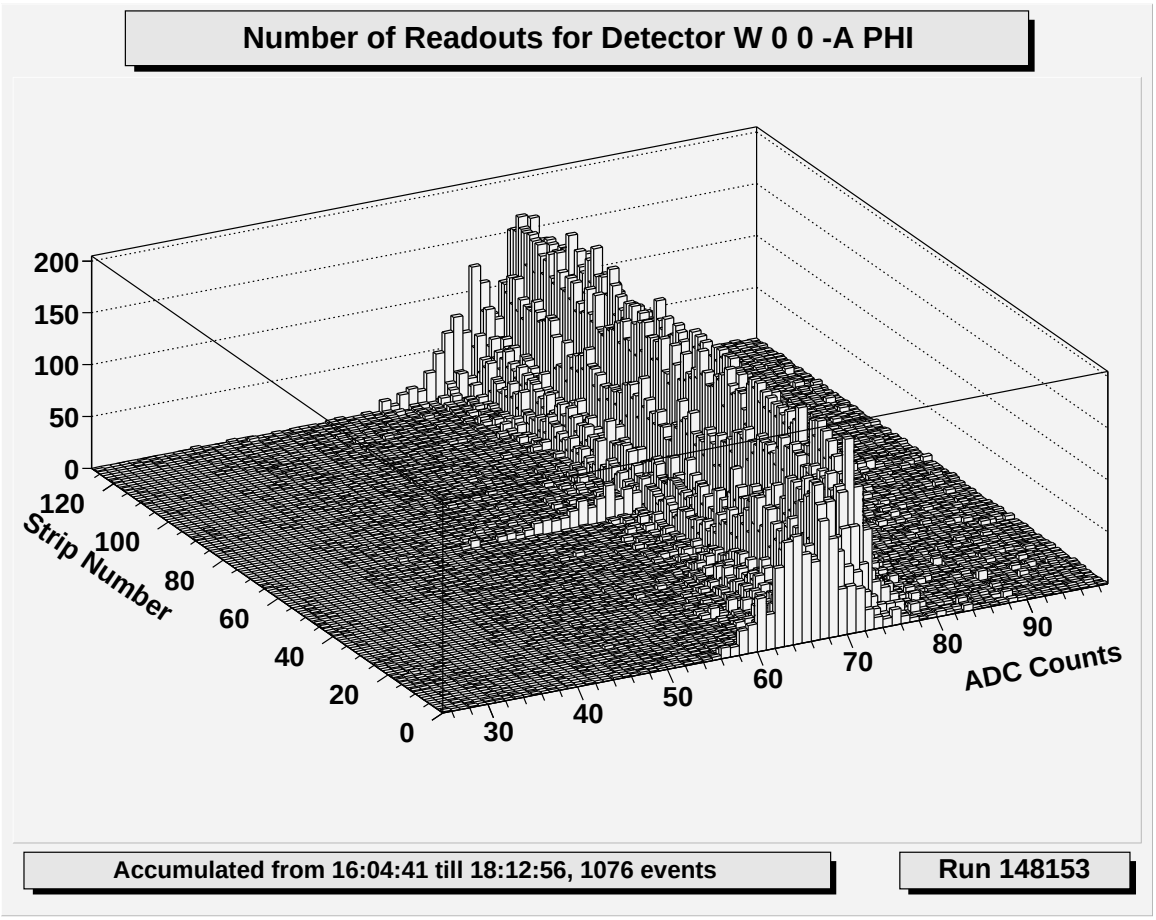


Figure 6: Example pulse height distribution for the L00 westmost sensor in wedge 0. The SVXMon command "histo strips2d adc {west west 0 0 phi} {nevents} 0" has been used to book this plot.

command arguments and their meaning is the same as for the **histo strips2d** command. The only exception is that the keyword "nhits" in the list of plot types *histoTypes* should be used instead of the keyword "nevents" in case the user wants to see the total number of channels read out. The command returns a handle which can be used to suppress one of the histogram bins. It works in a similar way to the handle returned by the **histo strips2d** command. Example **histo daqparam** plots are shown in Fig. 8 and 9.

Another useful DAQ-related plot can be booked with the command

histo bxtrace

This command takes no additional arguments. It creates a stripchart plot of the bunch crossing number vs. the event number which is useful for certain silicon DAQ studies. An example plot is shown in Fig. 10.

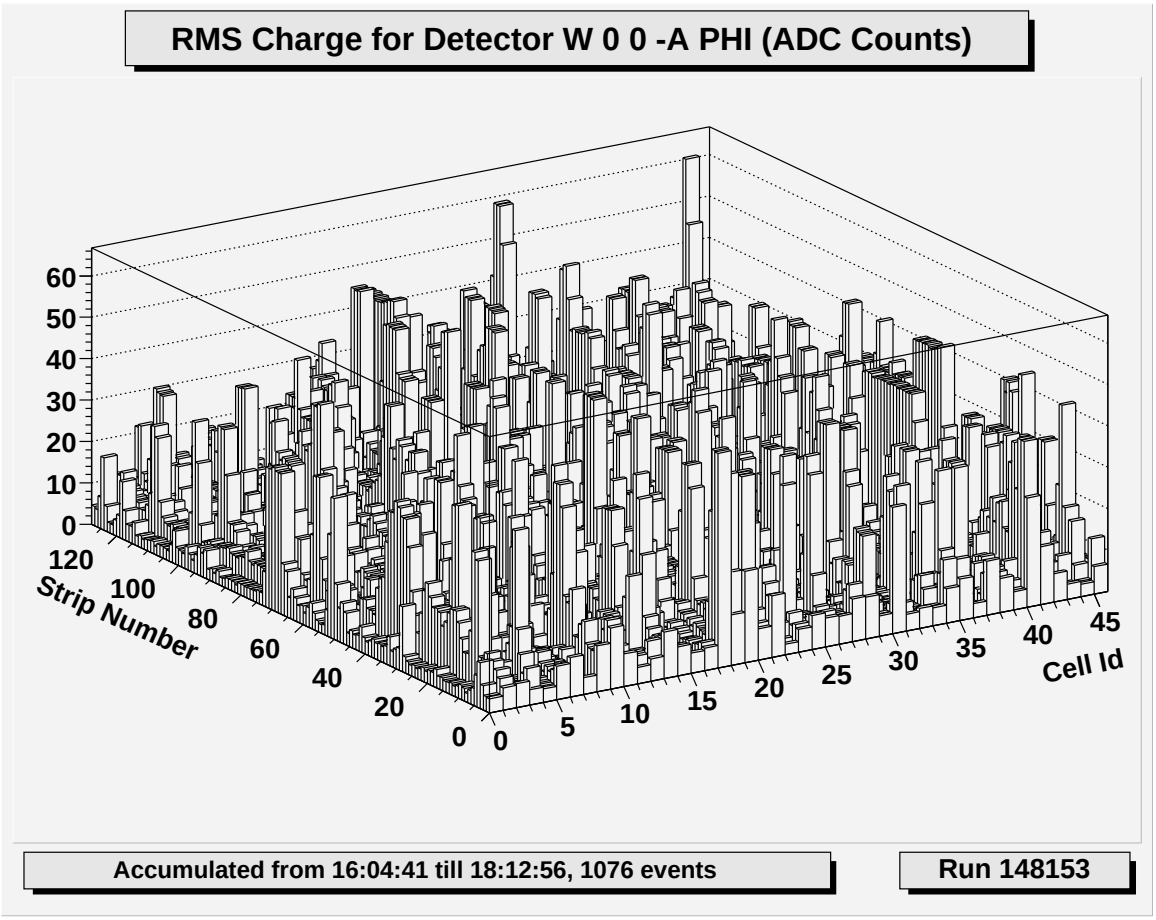


Figure 7: Example plot of RMS charge vs. the strip and pipeline cell numbers for the L00 westmost sensor in wedge 0. This plot has been booked with SVXMon command "histo strips2d cellid {west west 0 0 phi} {stdev} 0".

4.3 Chip Plots

Silicon quantities averaged over all good strips in each chip can be viewed on the plots created by the following command:

histo chips *histoTypes makeSnapshots*

This command creates several ROOT canvases which display a set of monitoring plots in a tabular arrangement. Each histogram shows some quantity of interest vs. the chip number on a particular barrel, layer, and sensor side. The *histoTypes* argument specifies the list of quantities to plot. The meaning of the keywords in this list is quite similar (but not identical) to the meaning of the corresponding keywords in the strip-level plots:

"occupancy" — Plots average strip occupancy for a given chip, in percent.

"vrbocc" — Plots average VRB (level 1) occupancy for each chip, in percent (the CDF silicon DAQ may require a configuration adjustment to provide an access to this information).

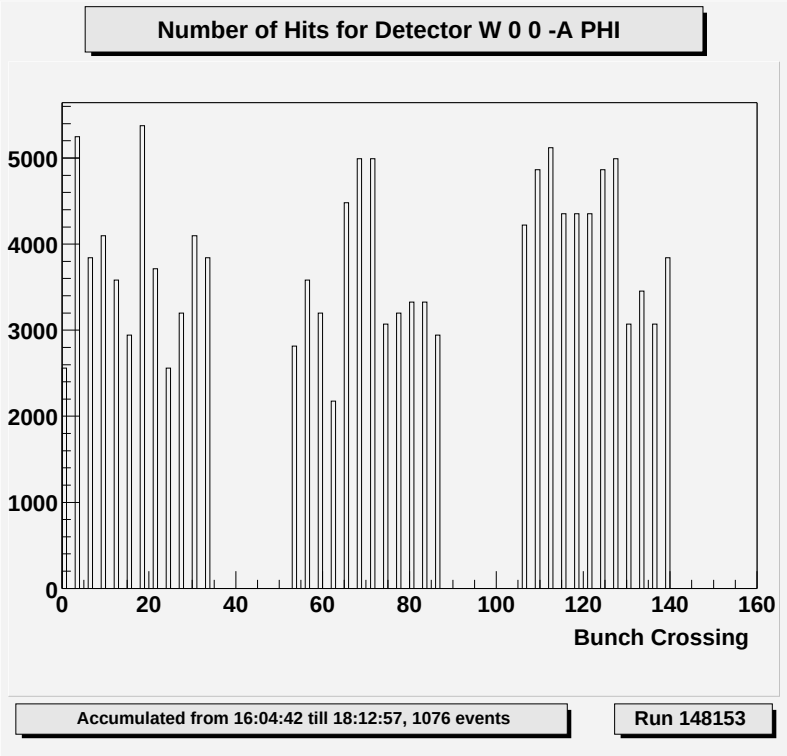


Figure 8: Example plot of number of channels read out vs. the SRC bunch crossing number for the L00 westmost sensor in wedge 0. Together with the plot on the next figure, it has been booked with the command "histo daqparam bunchx {west west 0 0 phi} {nhits mean} 0". Since L00 is used in "read all" mode, this plot essentially displays the distribution of bunch crossing numbers (up to a constant multiplication factor).

"nevents" — Plots the number of hits read out with a given chip. This plot differs from the occupancy plot only by normalization.

"mean" — Plots average pulse height for the hits read out with a given chip.

"stdev" — Plots the width of the pulse height distribution for the hits read out with a given chip. Note that this quantity is different from the average strip noise because it also includes strip-by-strip pedestal variations.

"skewness" — Pulse height distribution skewness.

"kurtosis" — Pulse height distribution kurtosis.

"bad" — Plots the number of bad strips connected to a given chip.

"discarded" — Plots the number of discarded strips for a given chip. See the description of "svx badstrips" command in section 7.4 for a discussion of bad and discarded strips.

"newbad" — Plots the number of strips which have the "bad" tag but don't have the "discard" tag. This plot and the next one may become useful in the future if real-time bad strip diagnostics is implemented in SVXMon.

"newgood" — Plots the number of strips which have the "discard" tag but don't have the "bad" tag.

"dnoise" — Plots the width of the distribution of pulse height differences between adjacent

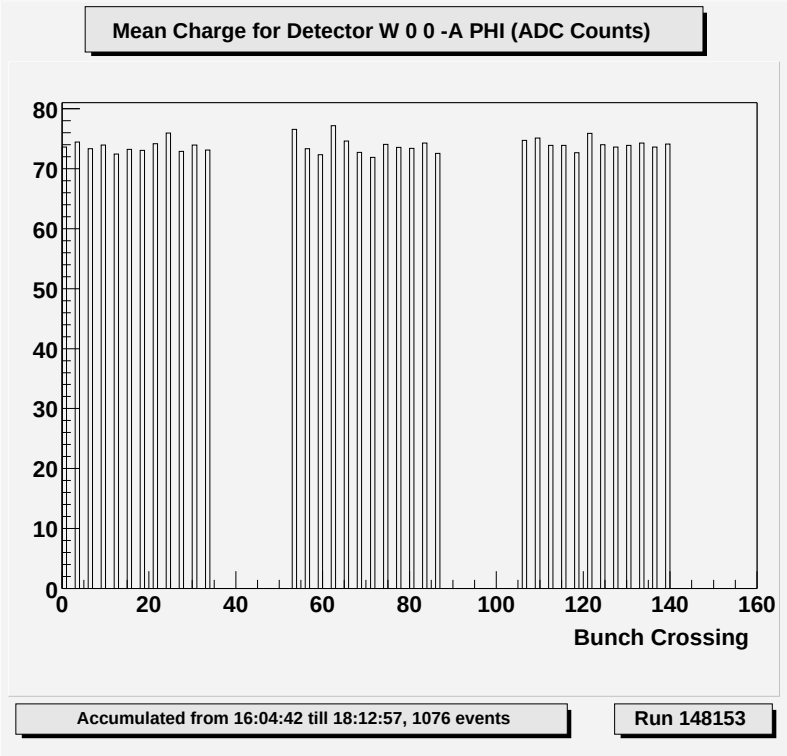


Figure 9: Example plot of mean read out charge vs. the SRC bunch crossing number for the L00 westmost sensor in wedge 0. Since the mean charge is relatively flat across all bunch crossings present in the run, we can deduce that the front-end preamp reset signal has been safely confined to beam gaps.

channels, divided by $\sqrt{2}$. This quantity is different from the average dnoise because it also takes into account strip-by-strip pedestal variations. This plot makes sense only when the whole silicon system is used in "read all" mode.

makeSnapshots is a boolean argument which should be set to 1 in order to build the short-term plots. These plots will be placed on separate canvases rather than overlayed on top of the long-term ones because the number of plots on a chip canvas is already quite high (up to 48). As it was already mentioned for the strip-level plots, the value of *makeSnapshots* argument should be consistent with the settings of SIXDMon parameters "snapshotStrategy" and/or "vrbOccupancySnapshots" (section 7.3).

Example chip occupancy, mean charge, and RMS readout charge plots are shown in Fig. 11, 12, and 13, respectively. In these plots, the chip numbers on the horizontal axis increase with increasing phi wedge number, and within each wedge chip numbers increase in the readout order.

4.4 Half-Ladder Plots

Plots of various silicon quantities averaged over all good strips on each half-ladder side may be booked with the following command:

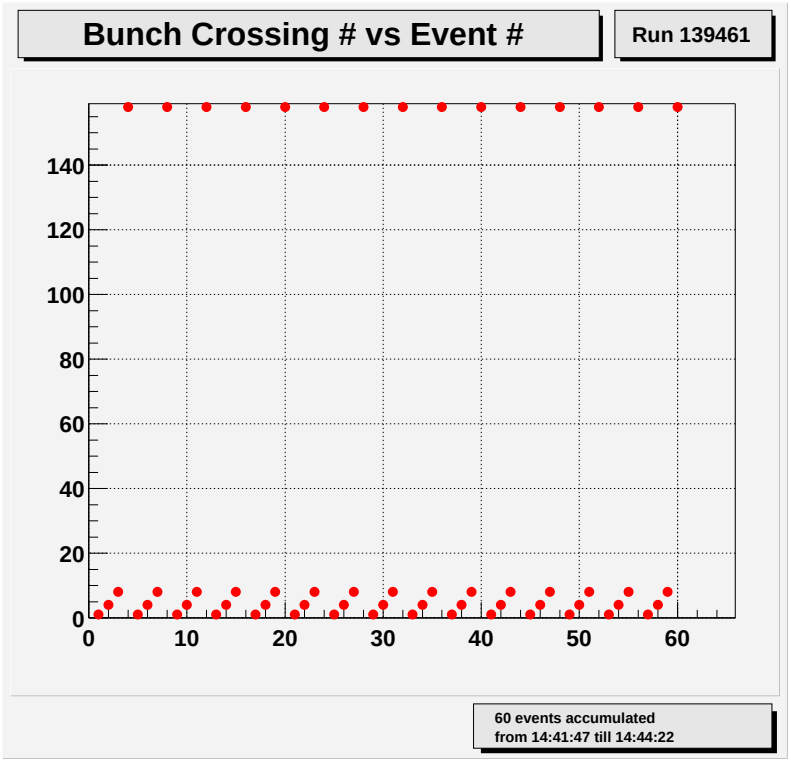


Figure 10: Example plot of the bunch crossing number vs. the event number.

histo ladders *histoTypes makeSnapshots*

The meanings of the arguments and the allowed keywords are exactly the same as in the **histo chips** command (section 4.3), except that the relevant quantities are calculated using all hits read out with a given half-ladder side.

The half-ladder plots are useful for looking up ladders included in the run and for a quick check of gross problems such as loss of readout by an HDI. However, corresponding chip-level plots seem to provide more useful information about the pulse height distributions because half-ladder plots incorporate additional smearing due to chip-to-chip variations. An example half-ladder occupancy plot is shown in fig 14.

4.5 Layer Plots

SVXMon knows how to combine pulse height distributions over layers in each detector half-barrel. The features of the aggregated distributions provide useful information about the dependence of occupancies, average readout charge, *etc.* on the distance from the beam and on the half-barrel position along the beam. Note, however, that pulse height distributions become significantly smeared due to ladder-to-ladder variations of pedestals and noise (this effect is more pronounced when the system is used without dynamic pedestal suppression on the front-end readout chips).

Three-dimensional plots of various silicon-related quantities on the z axis vs. the layer number on the x axis and the barrel segment number on the y axis may be booked with the following command:

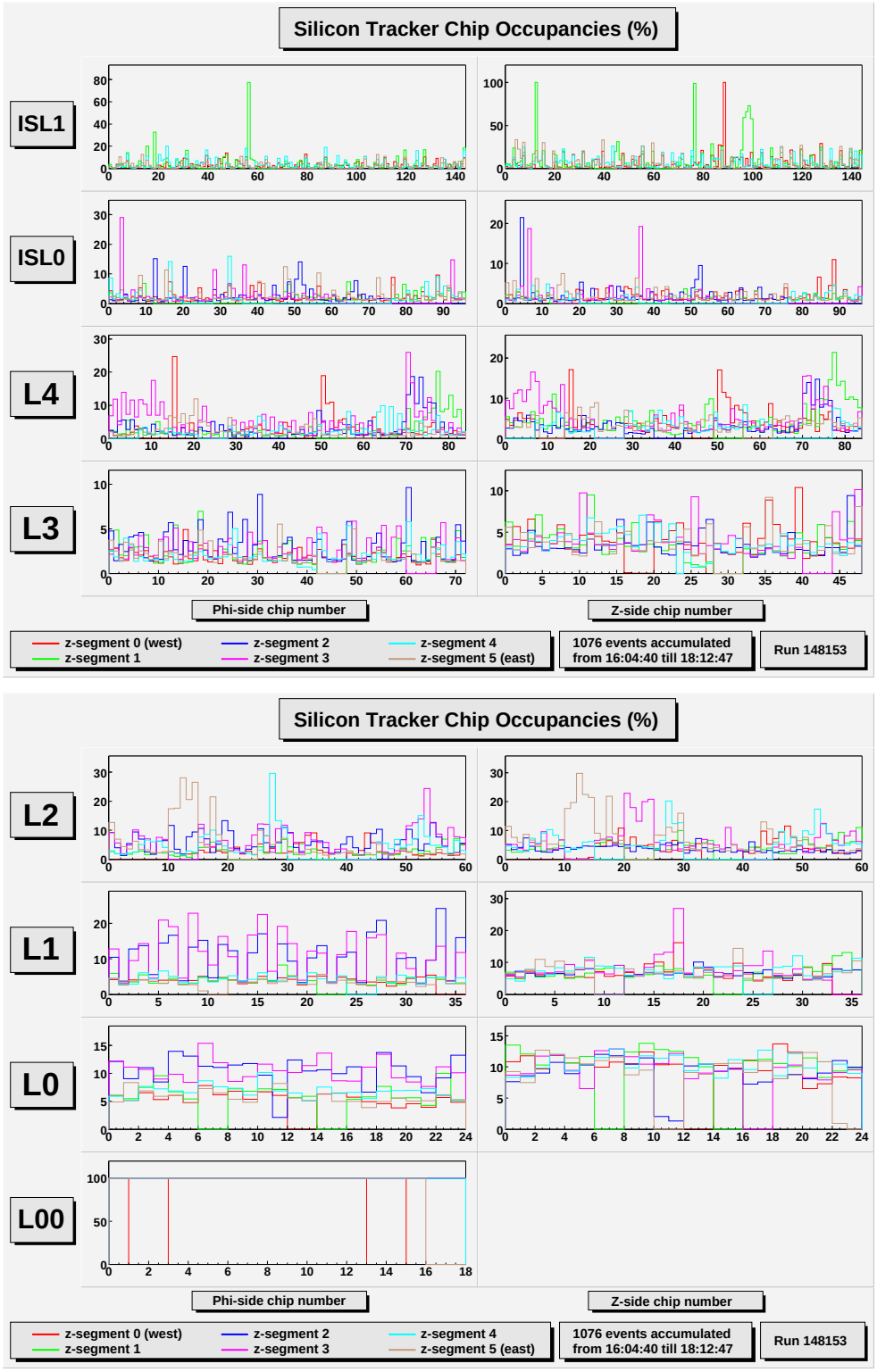


Figure 11: Example chip occupancy plots.

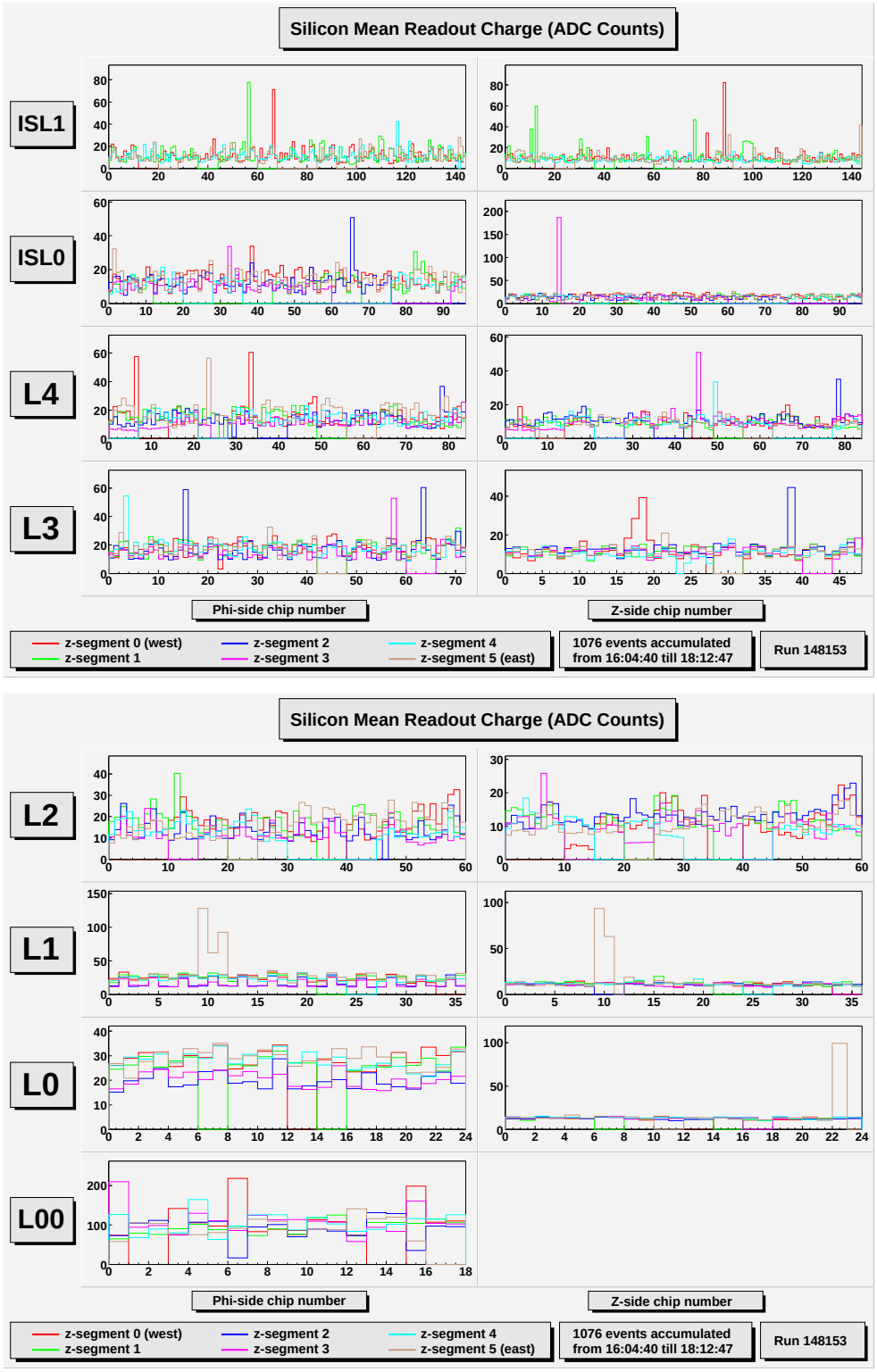


Figure 12: Example chip mean readout charge plots.

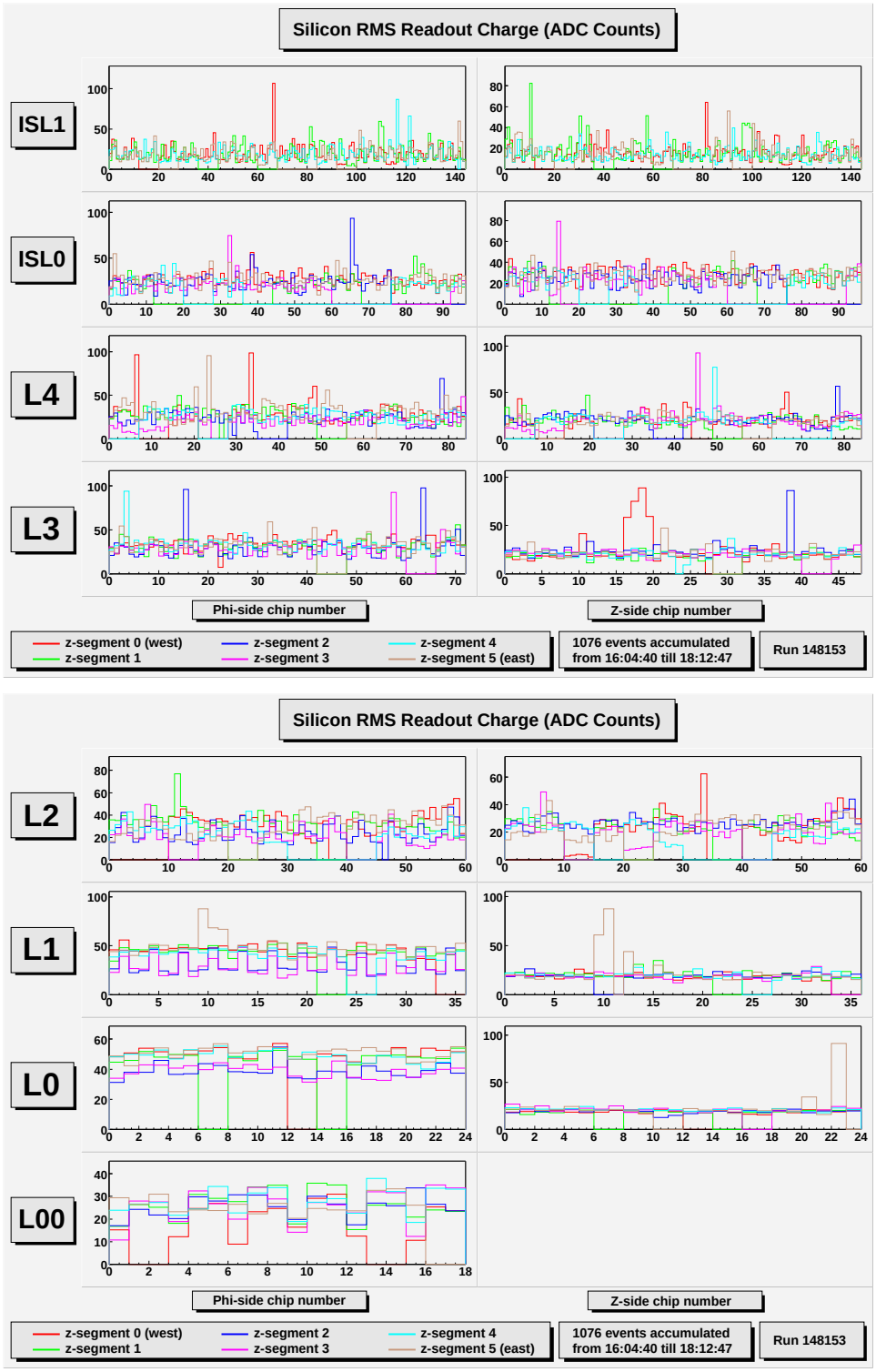


Figure 13: Example chip RMS charge plots. The RMS includes contributions from noise, signals, and pedestal fluctuations.

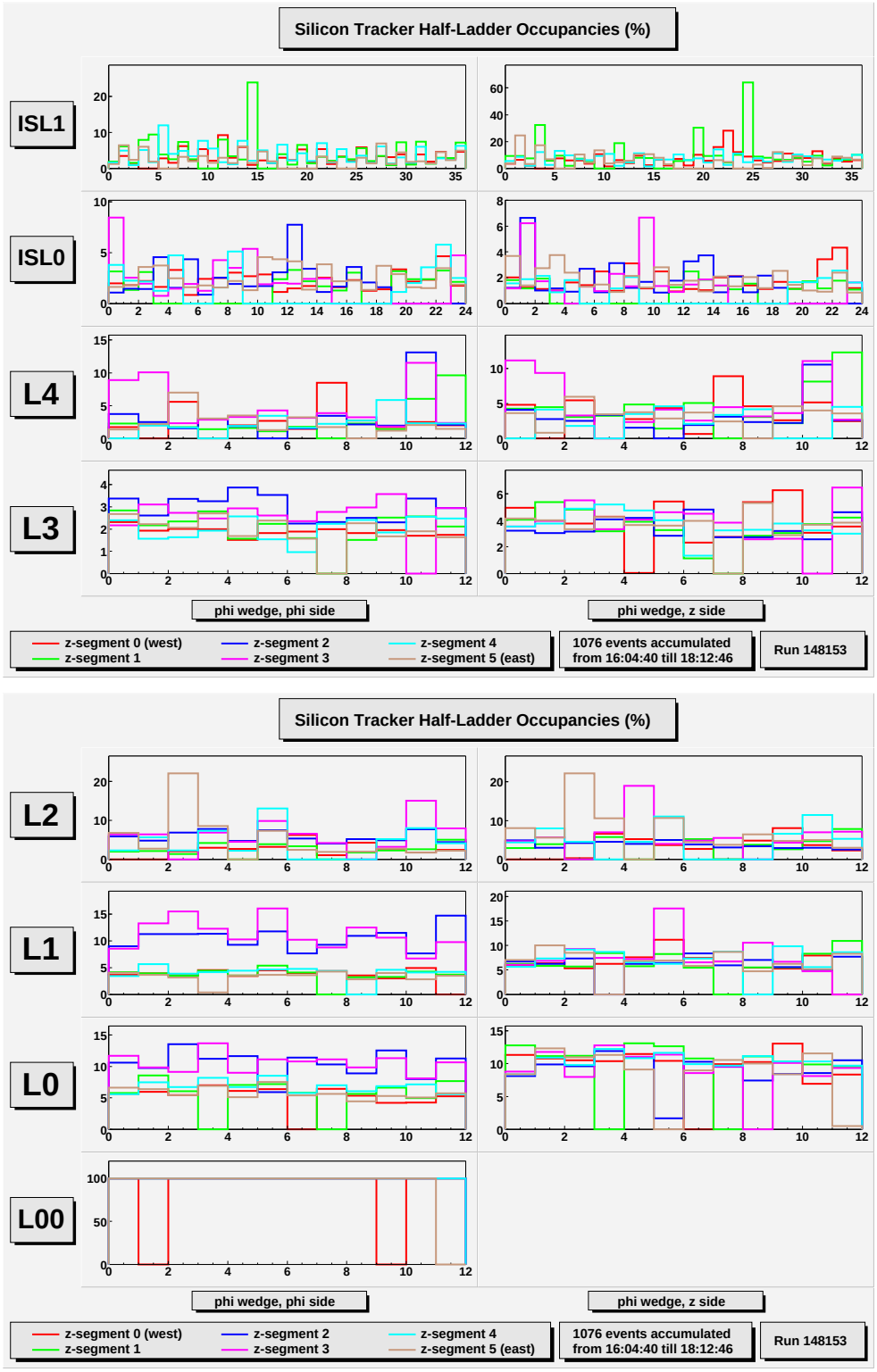


Figure 14: Example plots of half-ladder occupancies.

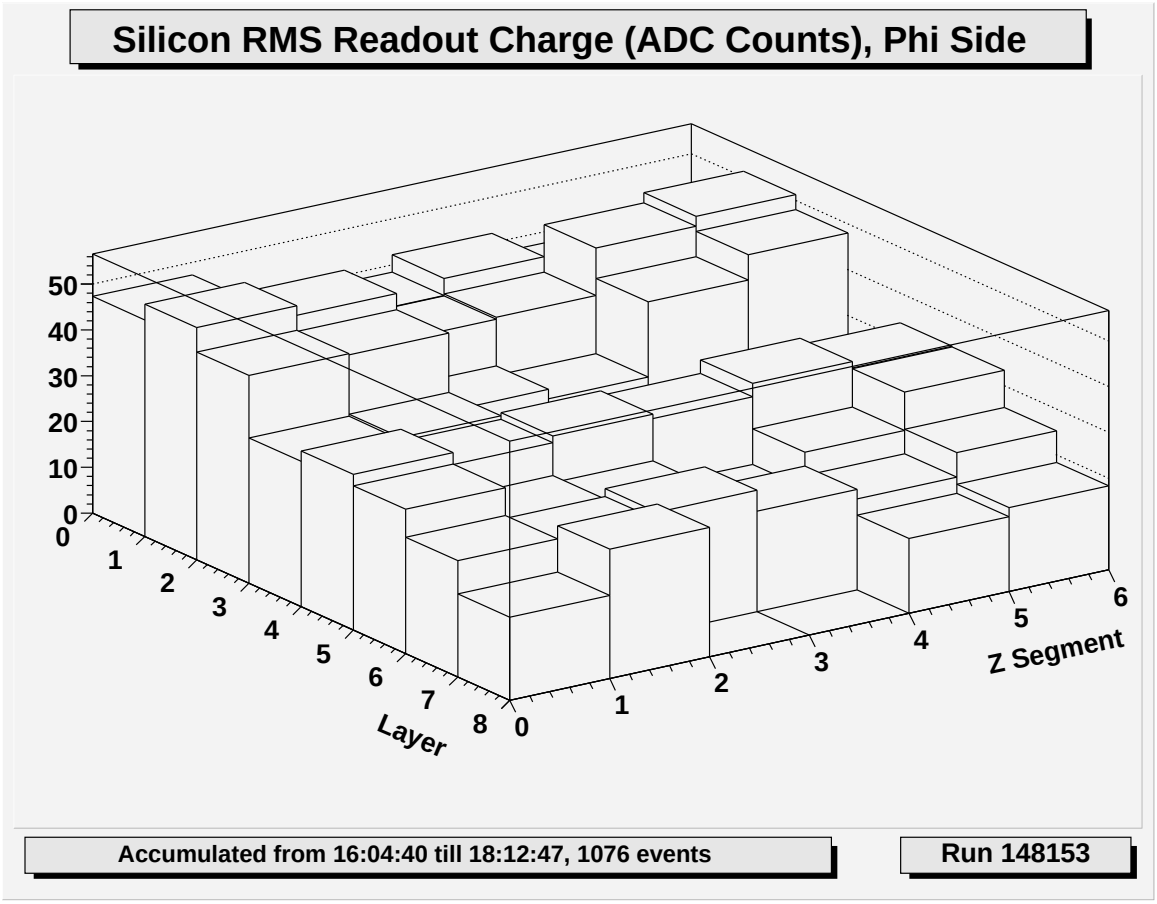


Figure 15: Example plot of the combined phi side RMS readout charge in each layer and half-barrel in the CDF silicon system.

histo layers *side histoTypes makeSnapshots*

where *side* is a string "phi" or "z". Arguments *histoTypes* and *makeSnapshots* have exactly the same meaning (and the same keywords may be used for *histoTypes*) as in the **histo chips** command (section 4.3), except that the relevant quantities are calculated using hits read out with all good strips which belong to a given silicon layer and barrel segment. You can see an example plot of this type in Fig. 15.

4.6 Occupancy Distributions

The plots of occupancy distributions are created in SVXMon by the command

histo occupancy *nbins xAxisType yAxisType snapshotStrategy*

This configuration command creates two canvases with long-term histograms of occupancies for each layer and barrel, as shown in Fig. 16. It also creates a canvas with occupancies aggregated over each side of L00/SVX/ISL which is illustrated in Fig. 17. Separate canvases for short-term occupancy plots may be created as well, depending on the value of

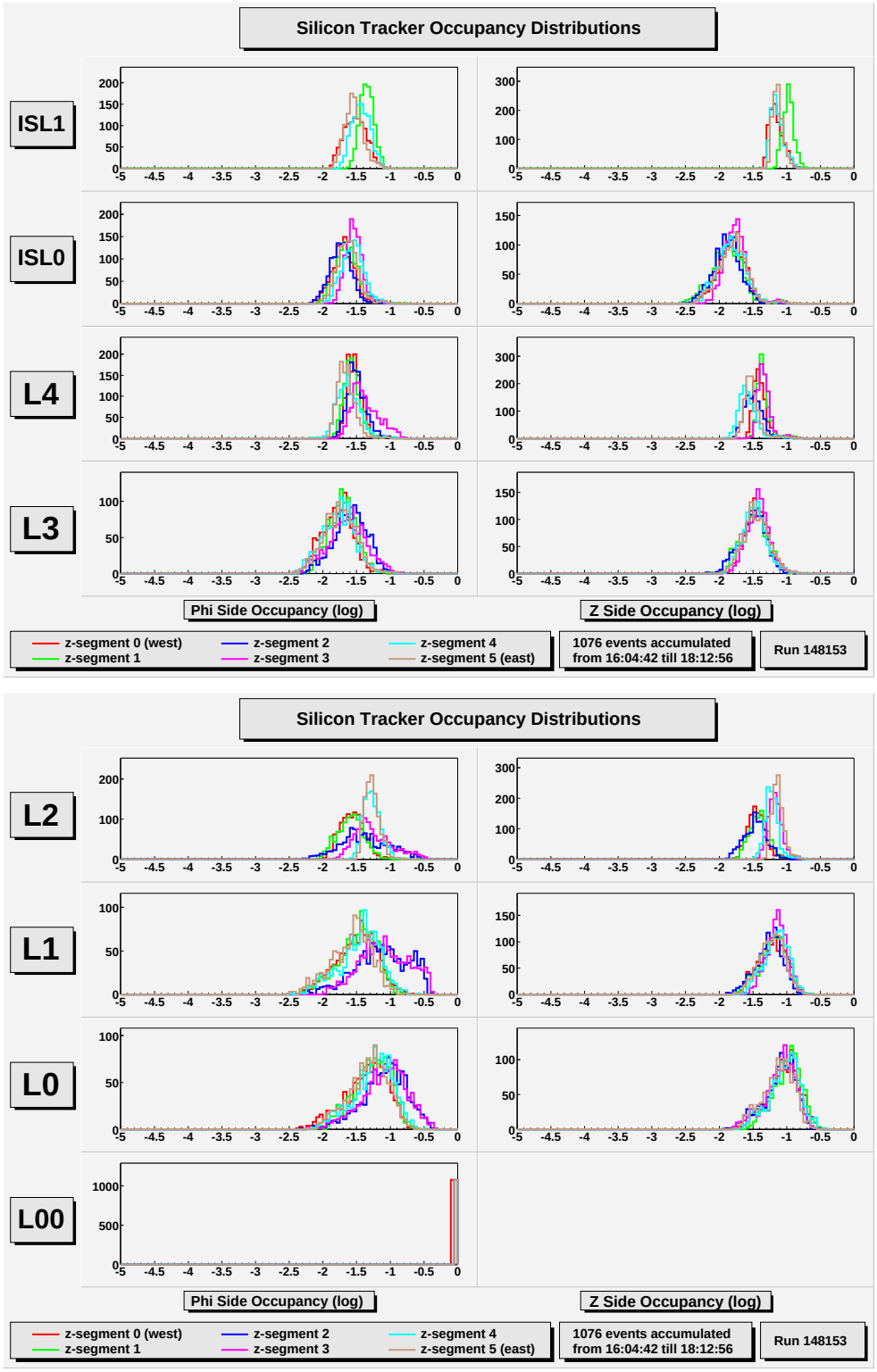


Figure 16: Example occupancy distributions.

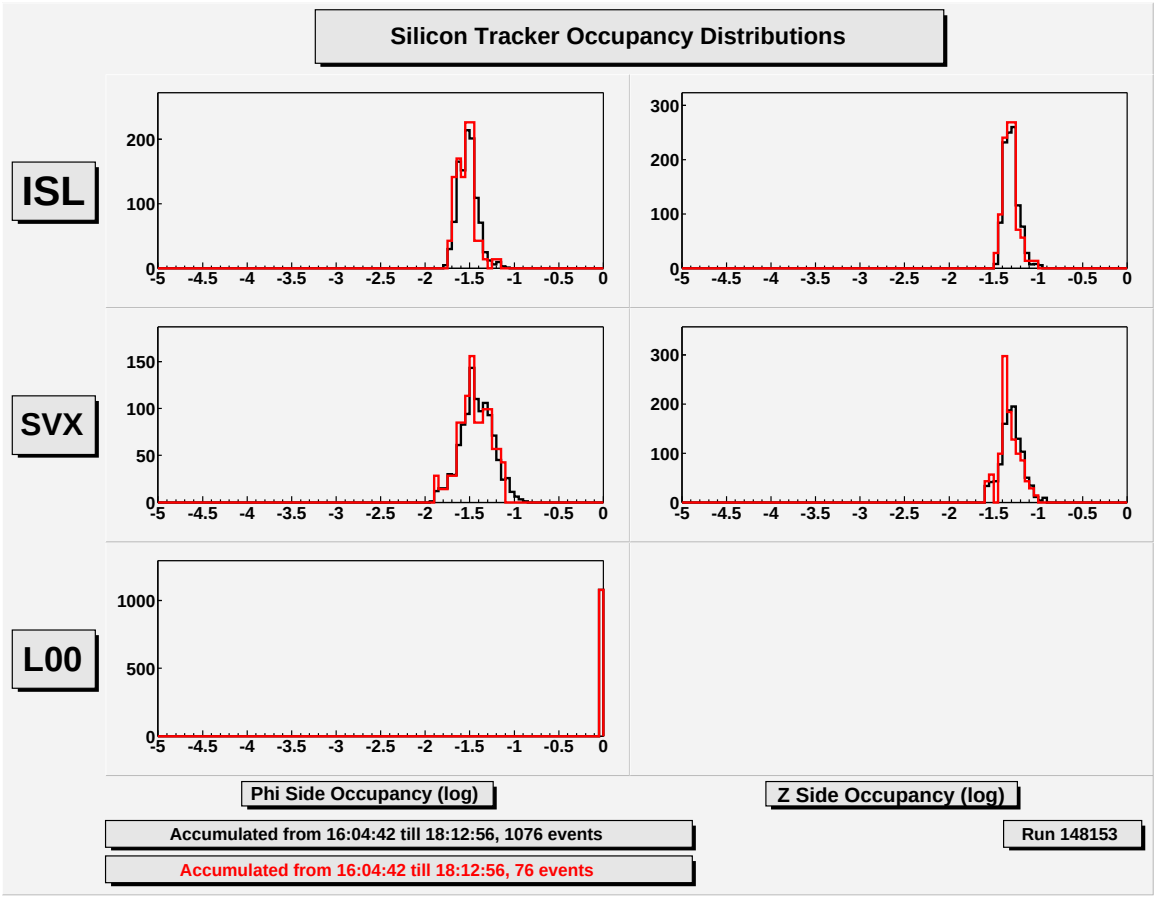


Figure 17: Example aggregate occupancy distributions for L00/SVX/ISL. In order to see the difference in the shape between the short-term and the long-term plots more effectively, the short-term histograms are normalized to the same number of events as the long-term ones.

the *snapshotStrategy* argument. The distributions shown are true occupancy histograms, one entry per event (all plots described earlier average occupancy values over many events). The plots only use hits on strips which are not suppressed by the "svx badstrips discard" command (described in section 7.4), but they are normalized to the total number of strips including the bad ones.

The command arguments have the following meanings: *nbins* is the number of bins to use along the x (occupancy) axis for all histograms. *xAxisType* and *yAxisType* are strings which specify how to draw histogram axes. Valid specifications are "log" (draws a base 10 logarithmic scale) and "linear" (draws a linear scale). *snapshotStrategy* is an integer which specifies how the short-term plots are updated. Set it to 0 if you don't want to create the short-term plots, to 1 if you want to use one snapshot, and to 2 if you want to interleave two snapshots. The snapshot updating schedule is specified for all histograms using monitor parameters "snapTimeSoftUpdate", "snapTimeHardUpdate", "snapEventsSoftUpdate", and "snapEventsHardUpdate" (section 7.3). Snapshots of the 2-d plots are *independent* from the global "snapshotStrategy" parameter of the monitor.

4.7 Tracking Plots

In order to increase the event processing rate, SVXMon does not attempt to run silicon tracking algorithms in a typical online monitoring configuration. However, it can plot certain quantities which characterize silicon tracking performance using the information provided by SVT and/or generated by silicon track reconstruction during data processing in the Level 3 trigger[2]. The tracking plots are booked in SVXMon with the following configuration command:

histo tracks *algorithmName*

This command creates plots which show the number of reconstructed tracks per event per half-ladder (track occupancy), average track residuals, and/or χ^2 for all sensors in the silicon tracker. *algorithmName* is an optional string which describes the algorithm used for track reconstruction. Valid algorithm names are "OutsideInAlg", "Regional", and "SVT". "OutsideInAlg" will be used by default in case *algorithmName* is omitted.

If the algorithm is not "SVT" then SVXMon must be configured to perform its own silicon tracking. To do that, one has to enable and configure modules `CT_Tracking`, `SiClusteringModule`, and `SiPatternRecModule`.

No matter which algorithm is chosen, the command creates two canvases devoted to SVT. One of them (illustrated in figure 18) shows the SVT track occupancy and the other displays the average χ^2 of SVT tracks (figure 19). All tracks found in the SVTD bank are used to build these plots, and no selections cuts are applied to these tracks.

More detailed plots are built for the tracks reconstructed with the selected algorithm. There are six canvases for each quantity (occupancy and residual/χ^2): two for each barrel (west, central and east), one showing layers L00 to L2, the other layers L3 to ISL2.

The **histo tracks** command returns a handle [9] which can be used for tuning of the plot behavior. Valid plot parameters which can be configured by the handle are listed below, together with their default values in parentheses:

nPhiHitMinG, nPhiHitMinB (3, 3)
: Two different sets of cuts can be defined: suffix B in parameter names is used for "bad" tracks, suffix G for "good" tracks. Cuts are made on the transverse momentum, p_T, and the number of z and phi side hits of the reconstructed track. Parameters **nPhiHitMinG** and **nPhiHitMinB** specify the minimum number of phi side hits for "good" and "bad" tracks, respectively.

nZHitMinG, nZHitMinB (2, 2)
: Minimum number of z hits.

ptHMinG, ptLMinB (0.5, 1.0)
: Minimum p_T.

ptHMaxG, ptLMaxB (500.0, 500.0)
: Maximum p_T.

debugFlag ("off")
: Turns debug printouts on or off.

[2] At the time of this writing, silicon tracking was not yet part of Level 3.

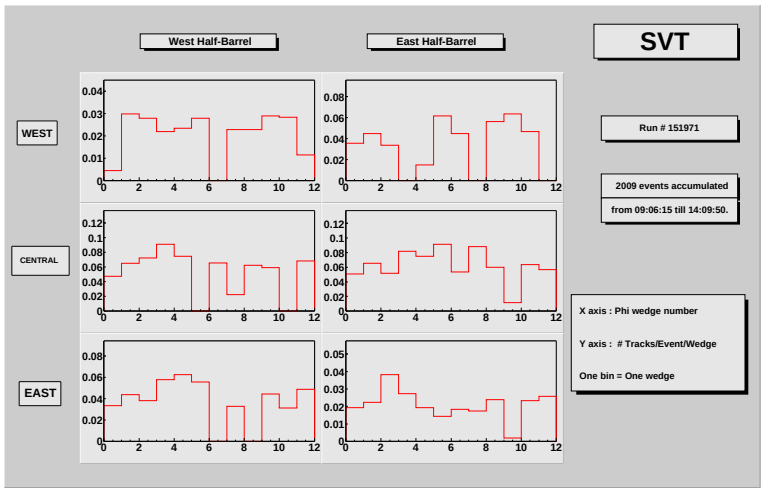


Figure 18: SVT Tracking Occupancy.

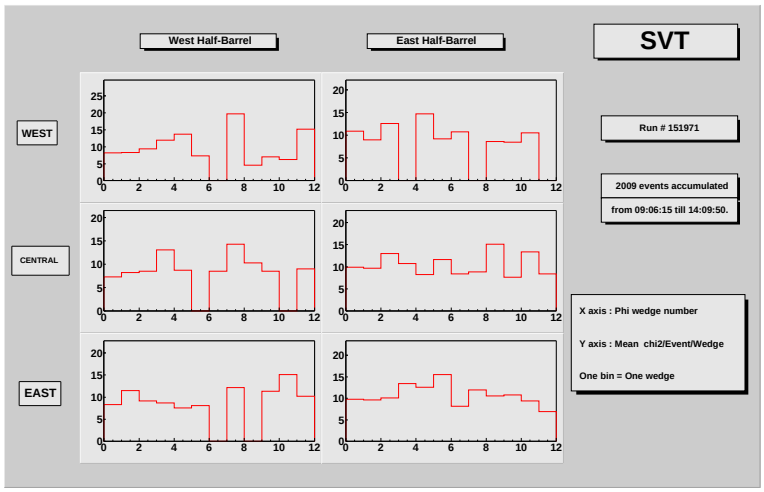


Figure 19: SVT Tracks average χ^2.

4.8 Pipeline Plots

The plots which provide information about the coherency of the silicon analog pipeline are created with the tcl configuration command `histo cellid`. Three different types of plots may be booked:

histo cellid global

Creates a histogram of the "correct" pipeline cell ids and turns the pipeline cell id mismatch monitoring on. The "correct" cell id is defined to be the cell id most often seen in the data stream. Although in theory this definition may fail to identify the "true" correct cell id in case of some pathological failure, in practice it is never a problem because the pipeline stays synchronous through a typical run on well over 50% of the front-end chips. Pipeline cell ids below 0 or above 45 should not be produced by a functional chip even if it is out of sync with the rest of the detector.

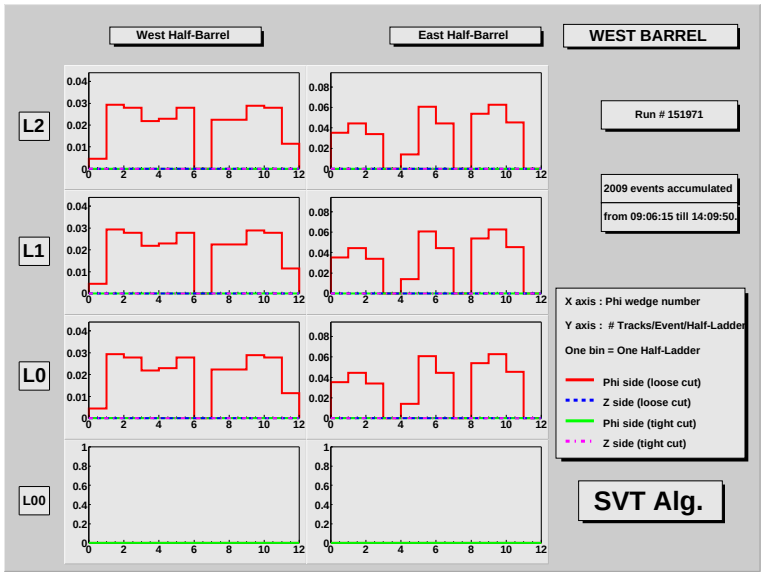


Figure 20: Example of Tracking Occupancy plot (here for SVT algorithm).

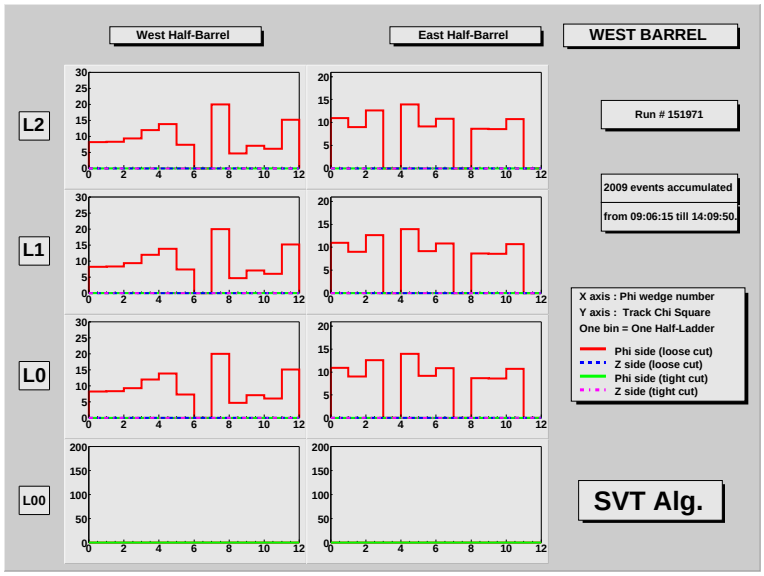


Figure 21: Example of track quality plot (here for SVT algorithm). When the SVT algorithm is chosen, this canvas shows the average χ^2 of the tracks. For the other algorithms, it shows the average residual of hits on tracks.

Such cell ids indicate some kind of a hardware failure, most often it is a sign of an optical transmission error.

histo cellid *detector chipNumSensor*

Creates the histogram of the cell ids encountered in the data stream for a given chip. The canvas also includes a plot of the difference between the chip histogram and the

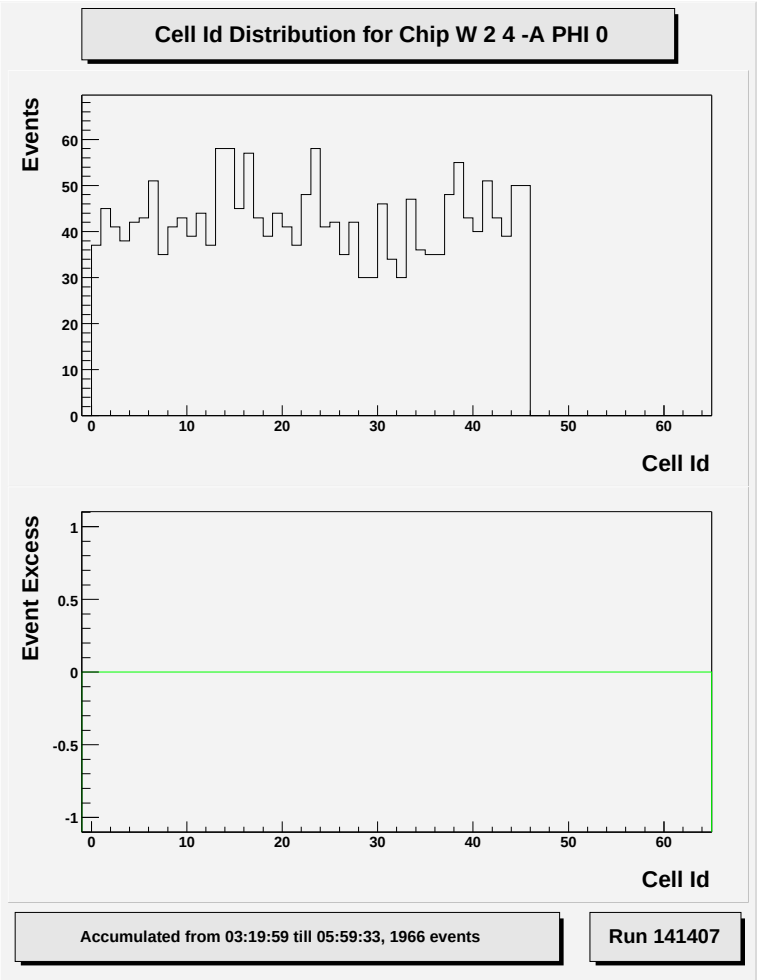


Figure 22: Example plot of chip cell id distribution. This chip's pipeline appears to be in sync with the rest of the detector.

histogram of the "correct" cell id. An example canvas is shown in Fig. 22. The command arguments have the following meaning: *detector* is the standard SVXMon half-ladder side specifier [8], and *chipNumSensor* is the chip number on the sensor side, in the readout order. Note that it is not necessary to call the **histo cellid global** command explicitly in case the cell id distribution is booked for at least one chip.

histo cellid map

This command creates a map in which silicon readout chips are color coded according to their pipeline status. Each chip on the map is shown with a small rectangle. The rectangle colors are associated with the pipeline status in the following way:

green — status is "good", pipeline is synchronized.

red — "bad", the chip has lost its synchronization (could also be a persistent optical transmission problem).

yellow — "no data", the bank unpacker was never able to read the pipeline cell id for this chip.

blue — "disabled", either a known problem or the chip was not included in the run.

The map is split between two canvases: one canvas is used for SVX and the other for L00/ISL. An example SVX map is shown in Fig. 23. This command also creates a histogram used to accumulate the distribution of the number of chips whose pipeline has lost synchronization. An example distribution is shown in Fig. 24. This distribution can be used to tune SIXDMon parameter "chipsNeededForHRR" (section 7.3).

The **histo cellid map** command returns a tcl handle [9] which can be used to tune the chip color coding algorithm. The algorithm tries to detect genuine pipeline desynchronization (which can happen, for example, due to a presence of noise on the front-end clock line), and to avoid tagging chips as bad due to spurious optical transmission problems. As such, the definition of the pipeline status has some built-in hysteresis: the status doesn't change until the difference between the number of events with incorrect and correct cell ids reaches a certain threshold.

The following parameters may be configured using the handle command (the numbers in parentheses show the default values):

hysteresis (3) — the value of this parameter specifies how large should be the difference between the number of events with incorrect and correct cell ids to warrant a change in the chip status.

needEvents (3) — the minimal number of events required to make a decision about the pipeline status. Until this number of events is accumulated, the chip status is set to "no data".

skipChip (0) — Array of boolean values used to enable or disable the chips on the map. The indices should look like this: skipChip($*detector*,$*chipNumSensor*). The value of 0 means chip is enabled. For example, the following code may be used to disable the chips on ladders which are not integrated in the current run:

```
# Create the cell id status map
set handle [histo cellid map]
# First, disable checks for all detectors
foreach detector [svx numerology all] {
    set nchips [svx numchips $detector]
    for {set chip 0} {$chip < $nchips} {incr chip} {
        $handle configure skipChip($detector,$chip) 1
    }
}
# Now, enable checks for the sensors included in the system
foreach detector [svx numerology [svx getrun]] {
    set nchips [svx numchips $detector]
    for {set chip 0} {$chip < $nchips} {incr chip} {
        $handle configure skipChip($detector,$chip) 0
    }
}
```

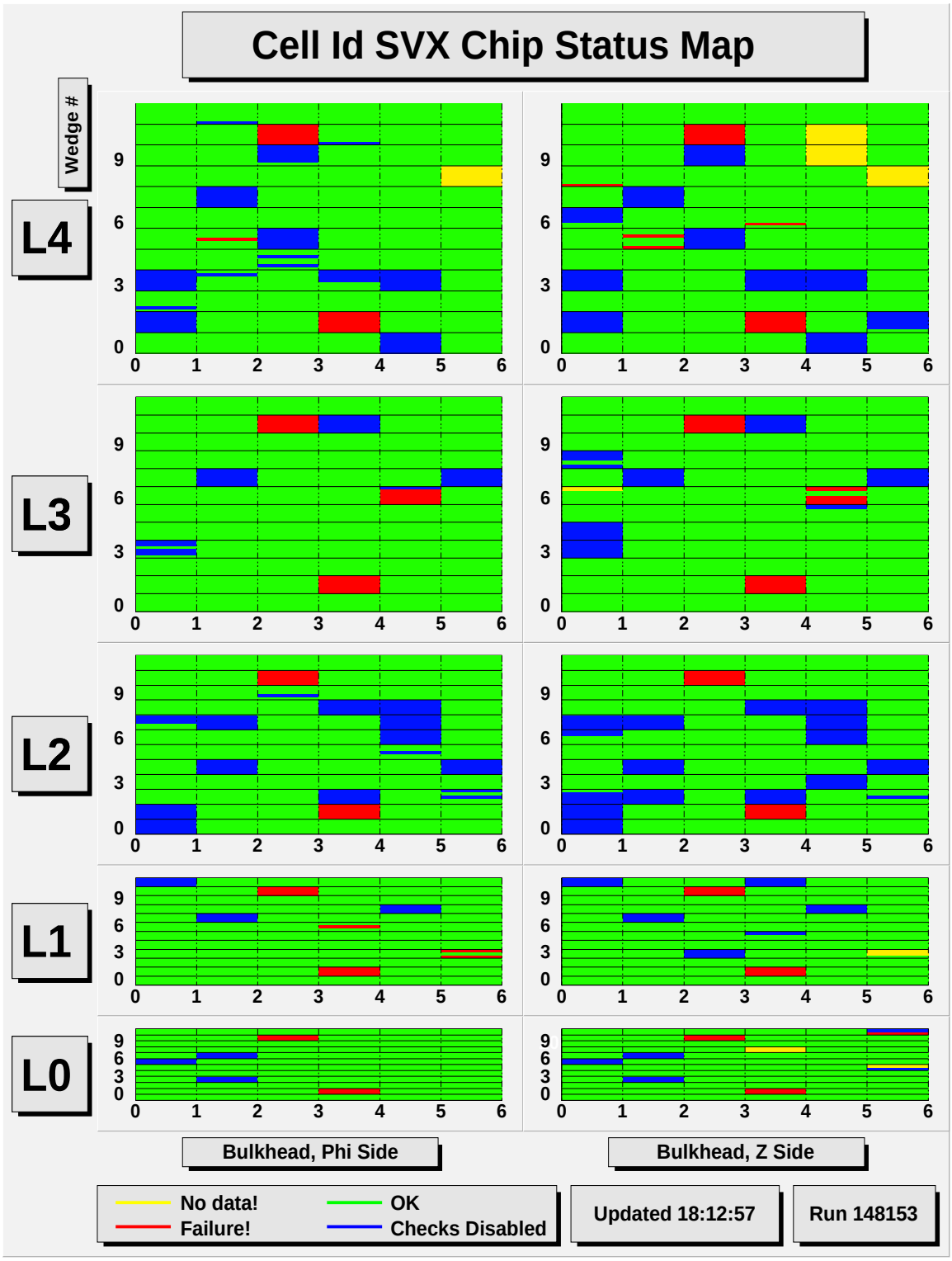


Figure 23: Example SVX pipeline status map.

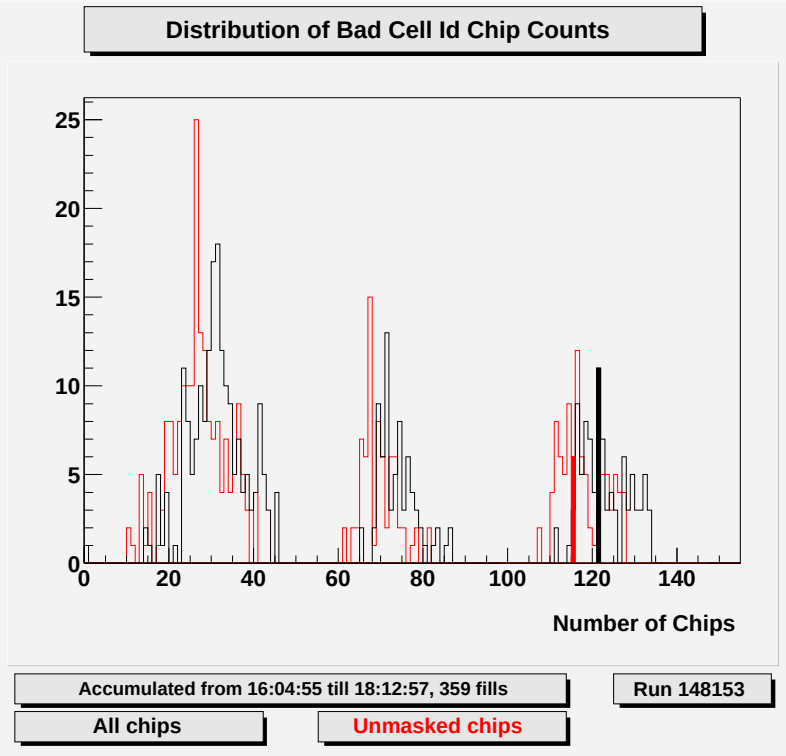


Figure 24: Distribution of the number of chips whose pipeline has lost synchronization. One entry is added to this histogram every time SVXMon plots are updated. The two solid bins are the ones most recently updated.

Other useful handle commands (besides setting and examining the parameters) are:

$handle chipStatus *chipSpecifier*
: Returns the current pipeline status for the given chip. One of the following keywords is returned: "disabled", "no_data", "bad", or "good". *chipSpecifier* is a two-element tcl list {*detector chipNumSensor*} in which *detector* is itself a tcl list which specifies the half-ladder side using the standard SVXMon notation [8], and *chipNumSensor* is the chip number on the sensor side in the readout order.

$handle chipEvents *chipSpecifier*
: Returns the number of events in which the unpacker was able to read the cell id for the given chip.

$handle numBadChips
: Returns a two-element list of integers {*n1 n2*} in which the first element is the total number of chips with pipeline status "bad" and the second is the number of bad chips which are not disabled on the pipeline cell map (this is the number of red rectangles on the map).

$handle numChipsSeen
: Returns the total number of chips for which the unpacker was able to read the cell id in at least one event since the beginning of the run.

$handle *chipType*

Returns a list of chip specifiers for various pipeline status types. *chipType* should be substituted by one of the following keywords: "badChips", "chipsSeen", "disabledChips", "extraEventsChips", and "missEventsChips". This command is mostly useful for debugging.

$handle updateConfig

This command can be used to force the immediate synchronization of internal object settings with the latest values of tcl configuration parameters. Useful for debugging purposes.

4.9 History Plots

SVXMon can monitor and display the time history of any of its long-term or short-term plots. The quantities shown on the history graphs can be values of select bins of monitored plots or values of various plot statistics (mean, median, standard deviation, *etc.*). The following command creates a history plot:

histo history *plotTitle listOfRecords*

The *plotTitle* argument specifies the title which will be displayed on the plot canvas. *listOfRecords* is a tcl list of history record identifiers. Each identifier has to be obtained beforehand by calling the "svx watch" command (section 7.4). The graphs for all history records in the list are overlayed on one plot. The number of elements in the list should not exceed 14 (this limit is, essentially, the number of distinct plot markers supported by ROOT). Example history plots are shown in Fig. 25. Please see the description of SIXDMon parameter "historyFrequencyDivider" in section 7.3 for details about history timing.

4.10 Periodic Plots

If we can assume that some silicon system parameter (*e.g.*, calibration charge during a gain scan) changes every N events then it is useful to build plots which show sequences of various silicon quantities averaged over N-event intervals. Such plots are created in SVXMon with the following command:

histo scan *scanName yBinType detector histoTypes*

Here, *scanName* is an arbitrary string which is used in the plot name and title in order to distinguish it from other similar plots. *detector* specifies one side of a silicon half-ladder in the standard SVXMon notation [8]. *yBinType* is a string which specifies how the quantity of interest will be presented. If *yBinType* is set to "strip" then the command will create a 2-d histogram with the cycle number on the x axis, strip number on the y axis, and the quantity of interest displayed along the vertical axis. If *yBinType* is "chip" then the chip number will be used on the y axis and the quantity of interest will be averaged across all good strips connected to each particular chip. The last possible *yBinType* value is "ladder" in which case the command will create a 1-d histogram, and the quantity of interest will be averaged over all good strips on the specified half-ladder side. *histoTypes* is a tcl list of histogram types to create. The histogram type determines which quantity will be displayed on the vertical axis. The following types are currently supported: "occupancy",

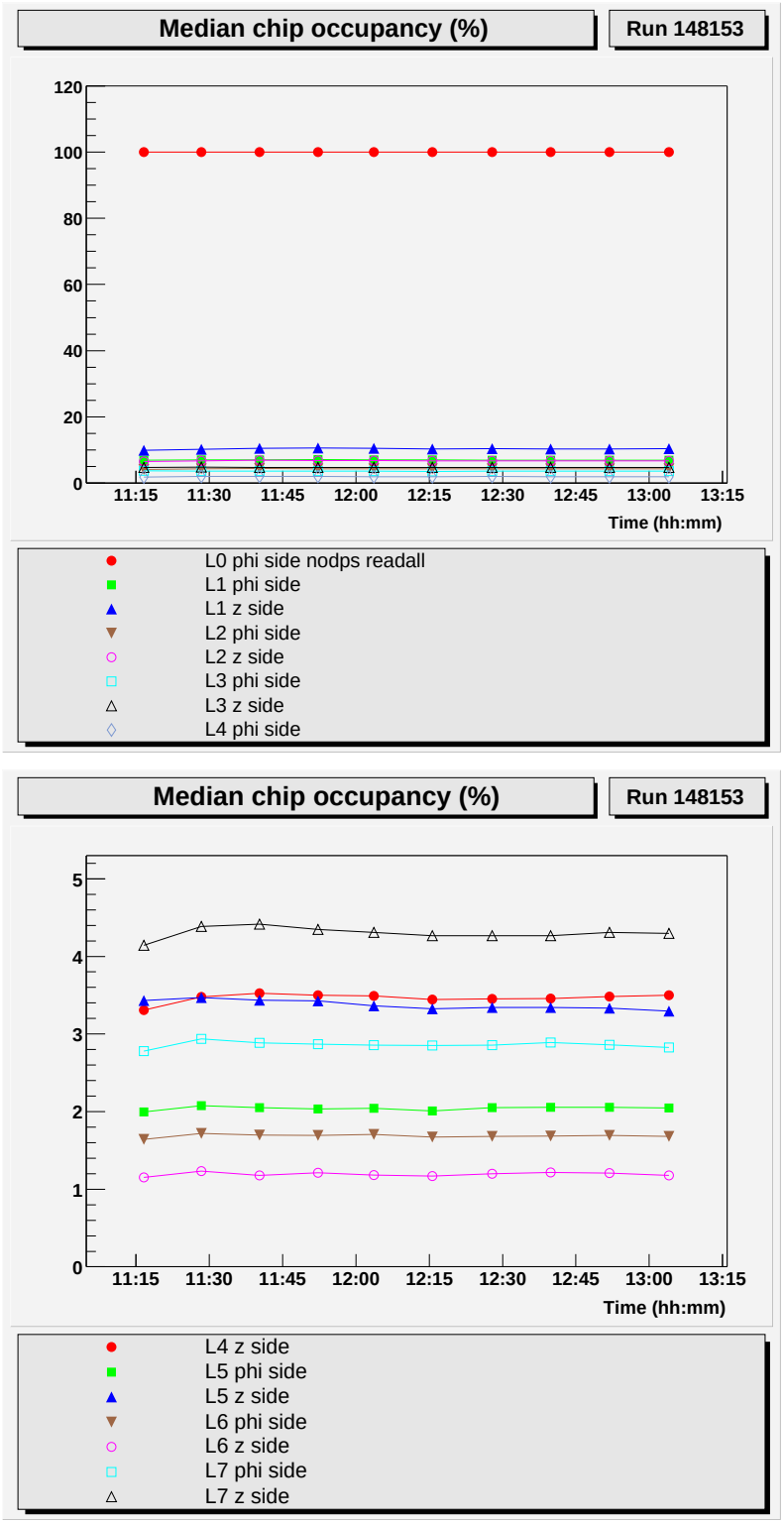


Figure 25: Example history plots of median chip occupancy by layer and side.

"nevents", "mean", "stdev", "skewness", "kurtosis", and "dnoise". Note that specifying "dnoise" as one of the elements makes sense only when the detector is used in "read all" mode. The command will create as many plots as the number of elements in the *histoTypes* list. The period N is defined simultaneously for all scan plots using SIXDMon parameter "scanPeriod" (section 7.3).

The **histo scan** command can only be used in an offline analysis because it delays the creation of ROOT plots until the end of job.

5 Online Quality Control

SVXMon can be configured to generate error messages when such silicon quantities as occupancies or pulse height averages are not consistent from chip to chip or deviate significantly from their expected values. The basic underlying assumption of the error checking algorithm is that most chips in the silicon system perform well, and the typical values of various quantities should be similar for chips with identical settings and capacitive loads. All chips in the detector are divided into groups based on their layer number, sensor side, dynamic pedestal subtraction setting, *etc.* (the program supports arbitrary aggregation of readout chips into groups). The median and range are estimated for the distributions of each quantity of interest (such as occupancy) over all member chips in the same group. The chips which deviate significantly from the group medians are reported to the error logger. The problems are also reflected on the chip status map plots. An example map of this type is shown in Fig. 26.

The automatic chip monitoring facility can be launched from either the configuration file or the SVXMon special prompt by the following command:

histo check *histoTypes makeSnapshots*

Here, *histoTypes* is a tcl list of quantities to monitor. Each element of this list must be chosen from the following set of allowed keywords: "occupancy", "vrbocc", "nevents", "mean", "stdev", "skewness", "kurtosis", "bad", "discarded", "newbad", "newgood", and "dnoise". The keywords have the same meaning as in the "histo chips" command (section 4.3). *makeSnapshots* is a boolean argument which specifies whether SVXMon can include short-term statistics in its determination of silicon data quality. Set it to 1 in order to use the short-term statistics.

The command returns a handle [9] which is used for tuning the analysis algorithm. The handle supports the following options:

$handle configure *paramName value*

Sets the value of *paramName* to *value* if *paramName* is a valid parameter name and *value* is an acceptable definition for this parameter. This command returns an empty string.

$handle cget *paramName*

Returns the value of parameter *paramName* if *paramName* is a valid parameter name.

$handle parameters

Returns the list of valid parameter names.

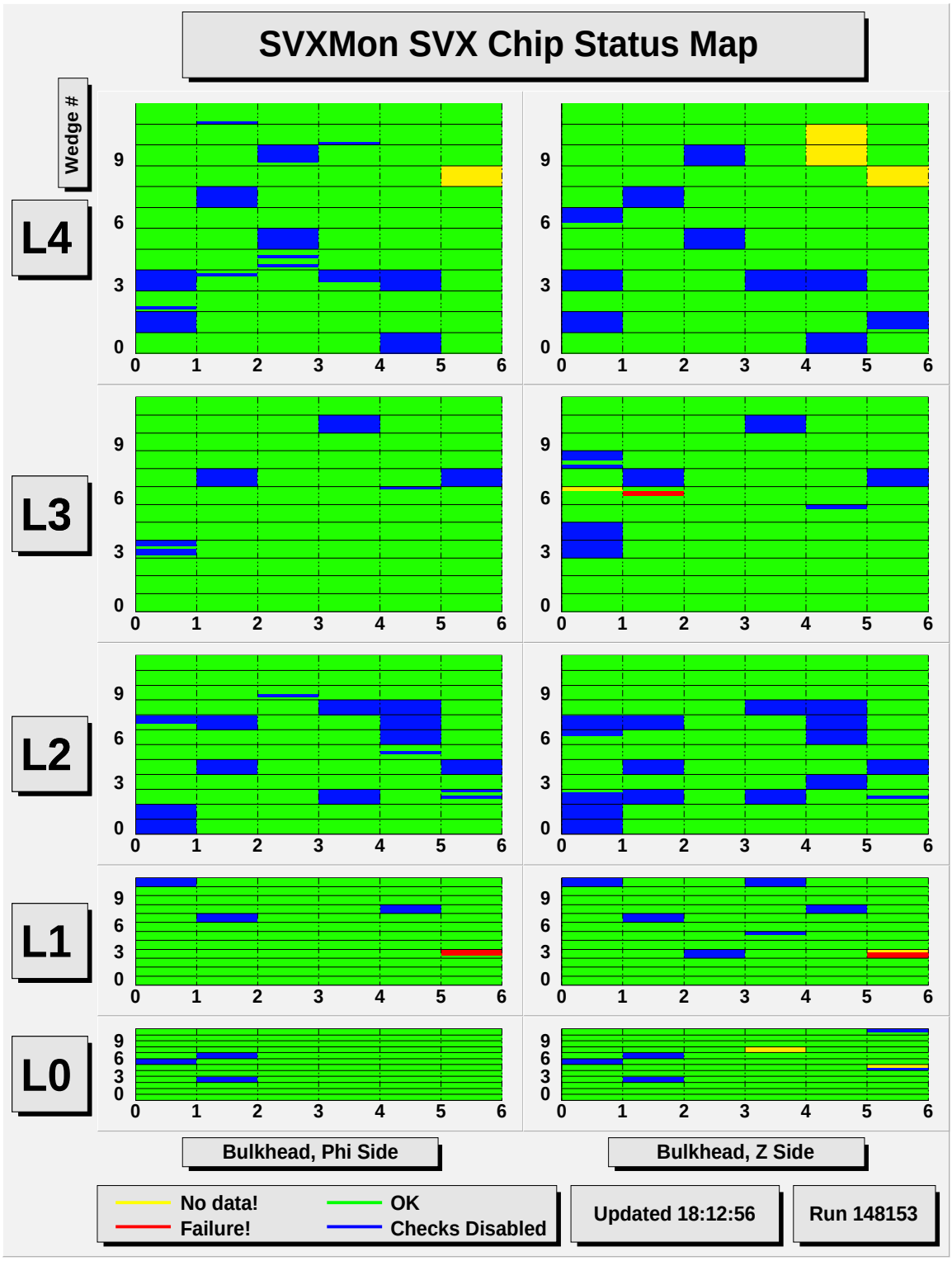


Figure 26: Example SVX chip status map.

$handle paramtable

Prints parameter values to the standard output in a tabular form.

$handle array *action paramName ?valueList?*

An optimized interface for array parameters. It is identical to the TclModule array interface described in detail in Ref. [11].

$handle chipgroup *action groupName ?chipList?*

This command is used to define, enable, disable, or view chip groups, depending on the *action* argument:

$handle chipgroup set *groupName chipList* — defines the set of chips which belong to the named group. The group name may be an arbitrary string, but it must not contain commas (because of the way group cuts are parsed — see the description of "groupcut" parameter later in this section). Each chip in the list of chips is specified as a list {*detector chip*} where *detector* is the five-element half-ladder side specifier [8] and *chip* is the chip number on this half-ladder side in the readout sequence. Each chip may belong to more than one group.

$handle chipgroup list — returns the list of group names defined so far.

$handle chipgroup state *groupName newState* — sets the state of the named group. The *newState* argument must be either "normal" or "disabled". The disabled groups will be ignored by SVXMon, and the program will not produce error messages if some group quantity goes out of limits. This command is useful in order to quickly suppress a flood of messages from a noisy or misconfigured chip group while SVXMon is running. When a new group is created by the **chipgroup set** option, it is always created in the "normal" state.

$handle chipgroup state *groupName* — shows the state of the named group.

$handle plotgroups

This command creates group plots of the quantities of interest, one bin per group. This command should appear in the configuration file just once, after all group names have already been defined.

The behavior of the monitoring algorithm depends on the values of the following parameters (default values are given in parentheses):

dynamicCuts (0)

The value of this parameter specifies which scheme will be used to determine allowed regions for various chip group statistics. 0 means use static cuts (defined by the **groupcut** parameter) and 1 means that the cut values will be determined dynamically every time a cut is applied by evaluating one of the **limitProcs** tcl procedures.

groupcut (0.0)

This parameter is an array of doubles. This array is used to define cuts for group quantities, such as chip occupancy medians. The indices of this array must be constructed as follows: `groupcut($cutName,$isSnapshot,$groupName)`. `$cutName` is a string which looks like "xxxxx_high" or "xxxxx_low" where xxxxx stands for the name of the quantity monitored (any histogram type keyword accepted by the "histo check" command). `$isSnapshot` must be 1 or 0. This parameter specifies whether the cut

will be applied to the statistic accumulated over a short period (when the parameter value is 1) or since the beginning of the run (when the value is 0). `$groupName` is the name of the chip group for which the cut is defined. If some cut is not defined explicitly in this array, it will be automatically set to 0. The cuts defined in this array will be used only if the **dynamicCuts** parameter is set to 0.

limitProcs ("proc dummyErrLimitProc {args} {list −1.0e12 1.0e12}")

This parameter is an array of strings which define tcl procedures used to calculate cuts on group quantities dynamically. The indices of the array are the names of the monitored quantities (any histogram type keyword accepted by the "histo check" command). Each procedure will be called with three arguments: $nEvents, $isSnapshot, and $groupName. It must return a two-element list of doubles {*$lower_limit $upper_limit*}. This mechanism allows for adjustment of cuts during the run depending on the number of events processed, elapsed time, luminosity, *etc.* The cuts defined in this way will be used only if the **dynamicCuts** parameter is set to 1.

chipcut (0.0)

This is an array of doubles used to define cuts for chip quantities, such as the chip occupancy and its deviation from the median occupancy in the group. The array indices look like this: `chipcut($cutName,$isSnapshot,$detector,$chip)`. `$cutName` is a string which must look like "xxxxx_limit_low", "xxxxx_limit_high", "xxxxx_nsigma_low", and "xxxxx_nsigma_high" where xxxxx stands for the name of the monitored quantity. Example:

```
set detector {west west 0 1 phi}
$handle configure chipcut(occup_limit_low,1,$detector,0) 0.5
$handle configure chipcut(occup_nsigma_low,1,$detector,0) 5
$handle configure chipcut(occup_limit_high,1,$detector,0) 10
$handle configure chipcut(occup_nsigma_high,1,$detector,0) 5
```

This sequence of commands instructs SVXMon that the snapshot occupancy of chip 0 on the given ladder [8] should be above 0.5% and below 10%, and it should not deviate from the median snapshot occupancy of the group by more than 5 "sigma". The "sigma" will be in fact determined in a robust manner as the interquartile occupancy range times 0.7413011.

minimalRange (0.001)

Array of doubles. In some degenerate cases (for example, occupancy in read-all mode) the group interquartile range of a monitored quantity may become 0 in which case it becomes impossible to define range multipliers which would allow chips with small deviations from the norm to pass the cuts. In order to be able to allow such chips to pass, we have introduced the lower bound on the range. The array indices should look like this: "xxxxx" (for run histograms) or "inst xxxxx" (for snapshots) where xxxxx is the name of the monitored quantity.

skipChip (0)

Array of boolean values used to enable or disable data quality monitoring chip-by-chip. The indices should look like this: `skipChip($detector,$chip)`. The value of 0 means enable checks. It is a good idea to disable checks for all chips not included in the run.

errorSeverity ("ELerror")

The array of strings which allows the user to change the error logger severity levels of various "Out of Limits" error messages issued when some group quantity gets out of limits. The indices look like this: "Xxxxx Out of Limits" or "Inst xxxxx Out of Limits", where xxxxx is the name of the monitored quantity (the first letter of an index is always in upper case). There are also several special indices: "No Data", "No VRB Occupancies", and "Wrong VRB Data Size". The valid severity levels are "ELincidental", "ELsuccess", "ELinfo", "ELwarning", "ELwarning2", "ELerror", "ELerror2", "ELnextEvent", "ELunspecified", "ELsevere", "ELsevere2", "ELabort", and "ELfatal".

chipSeverity ("ELerror")

The array of strings which allows the user to change the error logger severity levels of various "Out of Limits" error messages issued when some chip quantity gets out of limits. The indices look like this: "Xxxxx Out of Limits" or "Inst xxxxx Out of Limits", where xxxxx is the name of the monitored quantity.

maxEventsNoLadder (5)

The maximum number events which SVXMon will allow to pass before is starts reporting problems about chips not found in the data stream (after bank unpacking). Note that if the chip is included in the run but not found in the data stream, this doesn't automatically mean that there is a real problem. Instead, it could be a chip with high sparsification threshold in a quiet environment.

maxCallsNoLadderVRB (5)

The number of events in the VRB occupancy data must exceed this limit for SVXMon to start reporting that there is no VRB occupancy data for some chip (in case VRB occupancies are monitored at all).

needRunEventsMedians (10)

The number of events which SVXMon must process since the beginning of a run before reporting problems with chip group statistics.

needSnapshotEventsMedians (10)

The number of events which SVXMon must process before reporting problems with short-term chip group statistics. This parameter should be in agreement with the global snapshot schedule so that snapshots are not made more often than **needSnapshotEventsMedians** events.

needRunEventsChips (30)

The number of events which SVXMon must process before reporting problems with chip statistics accumulated since the beginning of the run.

needSnapshotEventsChips (30)

The number of events which SVXMon must process before reporting problems with short-term chip statistics.

padBottomMargin (0.25)

Double. The fractional bottom margin of the pad on which the group plots are placed. It is convenient to have this quantity as a configurable parameter because the optimal bottom margin depends on the length of group names.

labelSize (0.05)

Double. The fractional text size for the group names, as they will be displayed on the

plots of group quantities (one group name per bin). It is convenient to have this text size as a parameter because the optimal size depends on the number of chip groups defined.

All parameters except **padBottomMargin** and **labelSize** may be adjusted interactively in the middle of a run by executing the relevant commands at the SVXMon prompt.

In addition to the chip status maps, for each monitored quantity SVXMon creates a set of plots on twelve canvases which display this quantity together with associated upper and lower limits for each chip in the silicon system, one bin per chip. Example plots which display chip occupancy for the west side of the west barrel are shown in Fig. 27. Whenever some quantity goes out of limits, the color of the plot which contains the failing chip is changed to red. The chips with problems are marked with large vertical bars so that they are clearly visible, and an error message is sent to the error logger with the location of the failing chip and a description of the problem.

The chip group plots are generated per quantity monitored. An example group plot is shown in Fig. 28.

6 Offline Histogram Analysis

6.1 SvxMonRunCompare

The set of executables and scripts collectively called "SvxMonRunCompare" has been designed to monitor variations in the silicon system performance from run to run by comparing chip-level data. SvxMonRunCompare fills up and analyzes a ROOT ntuple which contains, for each run, several informative statistics for each chip. Such quantities as chip occupancy, mean charge, rms of the pulse height distribution, *etc.* are calculated from the strip-level plots (section 4.1) stored in the SVXMon root files. The chip settings needed to ensure similar chip configurations are extracted from the online database.

SvxMonRunCompare can be found in the the Run II software repository, package "SvxDaqUtils".

6.1.1 Ntuple Structure

Every entry in the SvxMonRunCompare ntuple represents a run. The ntuple contains a block with the run information and a block with the chip information for ladders included in the run. Tables 1 and 2 describe the contents of each block.

The global dynamic pedestal subtraction (DPS) setting in table 1 has the following meaning: -1 = undefined, 0 = DPS off, 1 = DPS on, 2 = mixed. For each chip, the occupancy is defined as the median occupancy of the 128 individual strips in the chip. The median is used rather than average in order to reduce the effect of bad strips. The chip mean charge and charge standard deviation are also defined as medians of the corresponding strip quantities. "stdevVar" is a more complicated variable which was chosen for its sensitivity to physics signal compared to noise. It reflects the variation of charge standard deviation from strip to strip. This variation is larger for chips which read out particle signals than for chips which read out only noise because the number of tracks passing through each strip and the deposited charge tend to exhibit non-uniformity which can't be attributed to noise

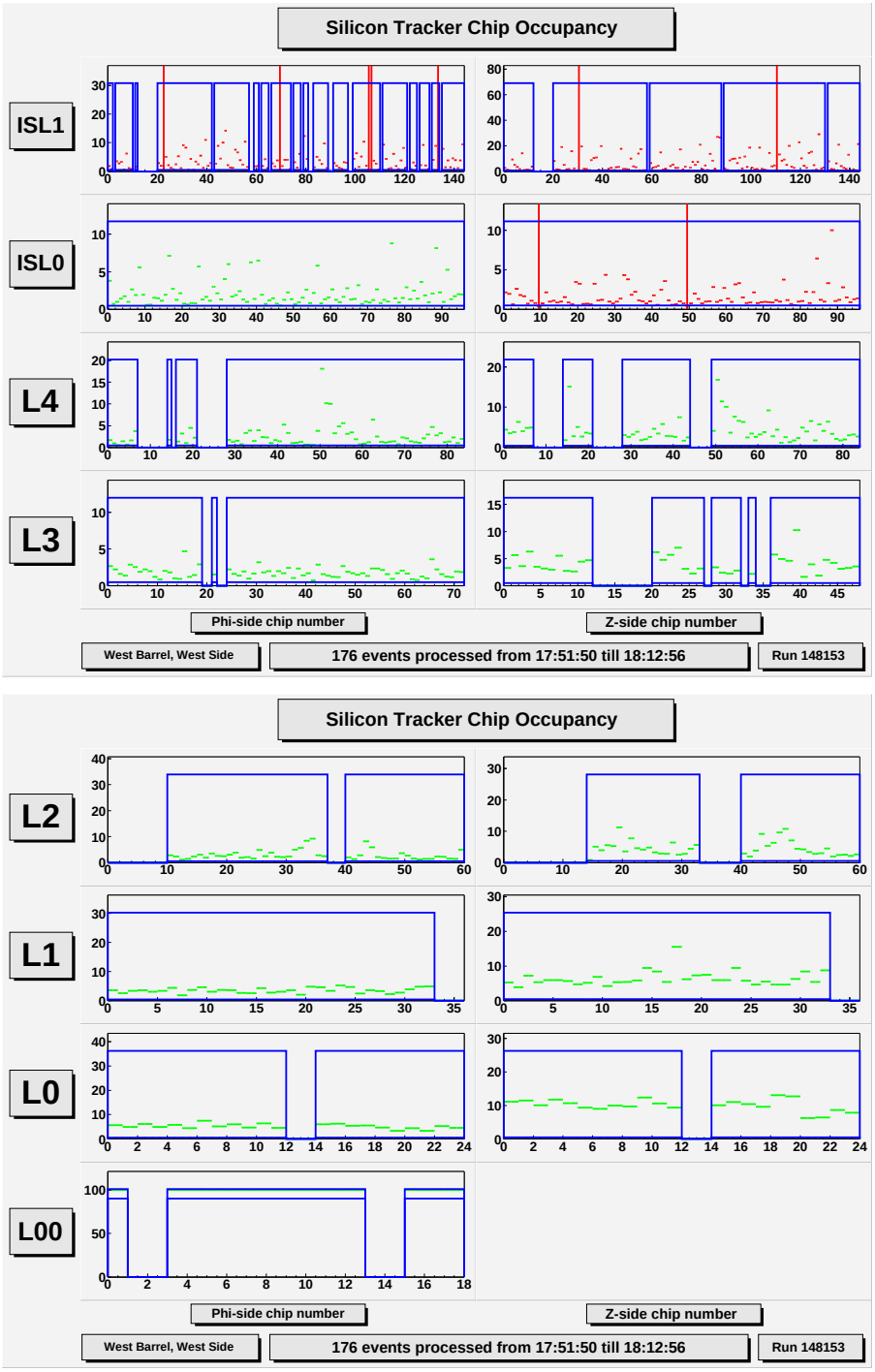


Figure 27: Example occupancy monitoring plots for the west side of the west barrel. The chip numbers on the horizontal axis increase with the phi wedge number. Within each wedge they increase in readout order. The monitored quantity is drawn in green color if everything is normal, or in red if there is a problem. The limits are shown in blue.

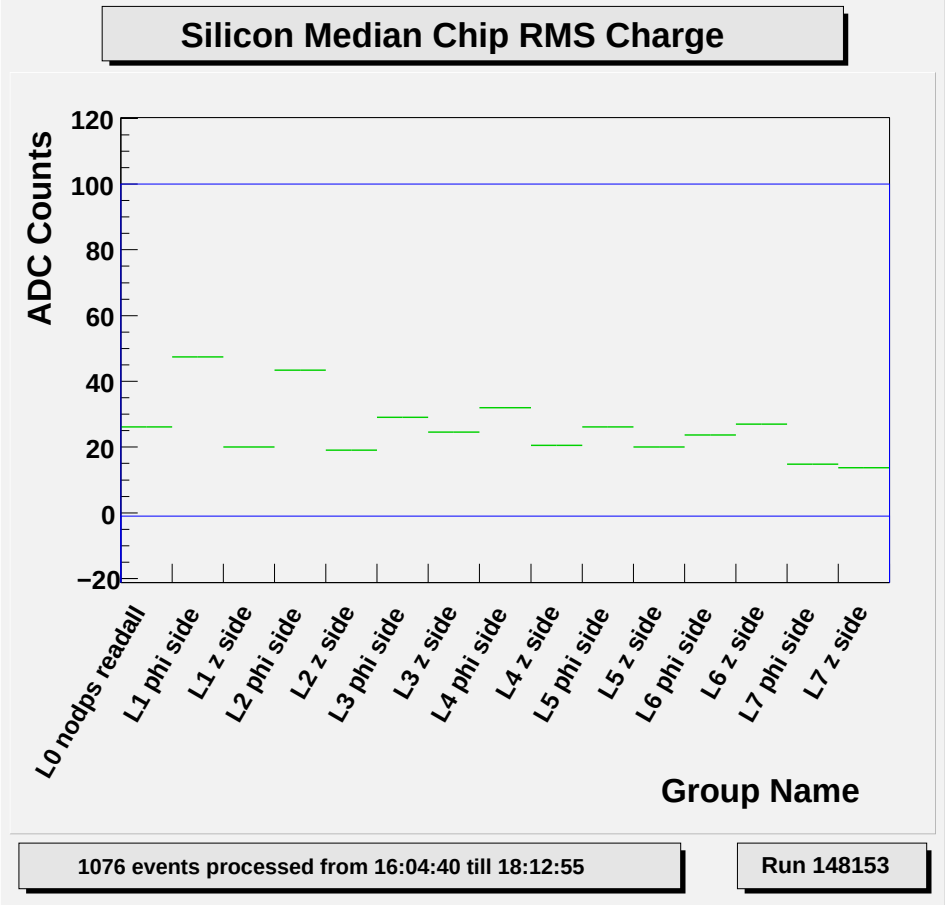


Figure 28: Example plot of median chip RMS readout charge. The chips are split in groups based on their layer, side, DPS mode, and readout mode (read all or sparsify). The values of the monitored quantity are shown in green, and the limits are displayed in blue.

alone. For each chip, stdevVar is defined as the median of $|\sigma_i - \sigma_{chip}|$ quantities, i=0,...,127, where σ_i is the standard deviation of the charge collected by strip i, and σ_{chip} is the charge standard deviation of the chip as defined above.

6.1.2 Run Comparison

In the course of normal online operation, SVXMon starts SvxMonRunCompare executables at the end of each run in its tcl configuration file (an example file is provided in appendix E). The number of events processed by SVXMon in this run should be sufficient for a reliable estimation of the chip statistics — as a rule of thumb, 200 events is enough. After appending the run to the ntuple (executable `appendSvxMonRun`), SvxMonRunCompare compares this run to a set of recent reference runs already present in the ntuple (executable `compareSvxMonRun`) and produces a log file and a ROOT file containing some interesting histograms. If necessary, it also generates alarms to the CO and the Silicon monitoring experts. A description of the run comparison executables is provided below.

Table 1: Description of the run block variables.

Variable	Type	Source	Description
runNumber	int	user	Run Number
version	int	hard-coded	Version of the Software
isAGoodRun	int	user	0 = Bad ; 1 = Good
nSvxMonEvents	int	SVXMon	Number of events processed by SVXMon
nChips	int	SVXMon	Total number of chips included in the run
runtype	int	database	Type of run as in Ref. [12]
nevents	int	database	Number of L3 events according to database
extra0	int	database	Run Control setting variable
modeGlobal	int	database	Global DPS setting
modeL00	int	database	L00 DPS setting
modeSVX	int	database	SVX DPS setting
modeISL	int	database	ISL DPS setting
rundate	int	database	Date of data taking

- appendSvxMonRun

  Usage:

  ```
  > appendSvxMonRun runNumber SvxMonFile TreeFile Option
  ```

  Here, "SvxMonFile" is the SVXMon histogram file corresponding to run "runNumber" and "TreeFile" is the ROOT file which contains the run comparison ntuple. appendSvxMonRun appends a run to an existing ntuple (option "update"), or creates a new ntuple and appends the run to it (option "create").

- compareSvxMonRun

  Usage:

  ```
  > compareSvxMonRun runNumber TreeFile ParameterFile LogDirectory \
                     -n numberOfRuns -m maxRun -p minRun -s scriptName
  ```

  `compareSvxMonRun` is the main executable of SvxMonRunCompare. It compares the run specified by "runNumber" to a set of reference runs, produces a text log file, a ROOT log file, and, if necessary, alarm log files and an alarm pop-up window.

  ParameterFile: file containing a list of parameters and configuration variables. An example of such a file can be found in appendix F.

  LogDirectory: directory in which the log files will be stored.

  Options -n, -m, and -p determine the set of runs used as reference runs. Option -n is incompatible with options -m and -p. If used, -m and -p have to be used together.

  -n : number of runs to be used as reference runs.

  -m and -p determine the range of runs to be used as reference runs.

  -s : a tcl file that produces a pop-up window when executed if SvxMonRunCompare decides to launch an alarm and if the "pop_alarm" parameter is set to "t" in the parameter file.

Table 2: Description of the chip array block variables (see text and Ref. [13] for details).

Variable	Type	Source	Description
barrel	short	database	West=0, Central=1, East=2
halfLadder	short	database	segment (a.k.a bulkhead, or half-barrel)
phiWedge	short	database	Offline wedge number
layer	short	database	Offline layer number (L00=0, SVX L0=1, ...)
side	short	database	ϕ side = 0, Z side = 1
chipNumber	short	database	Chip Number in the readout sequence on the half-ladder side
key	int	database	SiDigiCode key
dpsOn	short	database	DPS. 0=OFF, 1=ON
bandWidth	short	database	Integrator bandwidth (rise time)
brs	short	database	Bias Ratio Select: affects comparator and ramp bias currents
readNN	short	database	Read Nearest Neighbour mode
readAll	short	database	Read All mode
countMod	short	database	Counter Modulo: ADC counter limit
drs	short	database	Driver Resistor Select: sets the output driver resistors
ramPed	short	database	ADC Ramp Pedestal
pDepth	short	database	Pipeline depth
chipId	short	database	Chip Id (in the daisy chain)
threshold	short	database	Threshold
lastChip	short	database	Set to indicate that this is the last chip in the daisy chain
fePol	short	database	Front End Polarity
rOrder	short	database	Read Out order
calDir	short	database	Cal Direction: charge injection polarity
rampDir	short	database	Ramp Direction
compDir	short	database	Comparator Direction
iSel	short	database	Bias Current levels
rampTrim	short	database	Ramp Slope Trim
mean	float	SVXMon	Chip mean charge
occup	float	SVXMon	Chip occupancy
stdev	float	SVXMon	Chip charge standard deviation
stdevVar	float	SVXMon	(see text)

SvxMonRunCompare starts the comparison by choosing a set of reference runs. It selects good runs (isAGoodRun != 0) for which SVXMon processed more than 200 events. With the -n option, it selects the last "numberOfRuns" such runs; with the -m -p options, it selects all such runs in the range [maxRun, minRun].

For each chip present in the run to be compared and for each of the variables "occupancy", "mean", "stdev", and "stdevVar", the program computes the average and rms of the variable over the reference runs, using only the runs in which the chip settings are exactly identical to the settings in the compared run. The deviation in the compared run from the average value is computed and normalized to the rms. In the subsequent description we refer to this normalized deviation as the significance of the variable, $\sigma_{variable}$. Figure 29 shows the distribution of the four variables for the reference runs together with their values in the compared run.

Diagnostics can be made based on the value and the significance of the four variables. Currently, three different checks are made:

1. "Chip with failure":

   A chip is said to have a failure if any variable value or variable significance is out of the following acceptable ranges:

   limit_low < value < limit_high
   nsigma_low < significance < nsigma_high

   where the cuts have been set in the SvxMonRunCompare parameter file (see appendix F). In case of a failure, the chip is reported in the text log file and in the ROOT log file (with a canvas such as the one shown in figure 29).

2. "Unbiased chip":

   By "unbiased chip" we actually mean a chip whose ladder is unbiased. Such a chip is characterized by a higher occupancy value, and lower mean, stdev, and stdevVar values than a biased one. Therefore, we tag a chip as unbiased if

   $\sigma_{occupancy}$ > bias_signif_high AND any other variable significance < bias_signif_low

   Currently, the cut values are set as follows: bias_signif_high = 6 for occupancy and bias_signif_low = -4 for all other variables.

   Any unbiased chip is reported in the text log file and the ROOT log file, and triggers an alarm log file and a popup window on the CO monitors.

3. General data quality cut:

   Figure 30 shows the occupancy significance for all chips present in the compared run. In this particular run, the distribution is well centered and its rms is close to 1 which means that there is no global discrepancy with the reference runs. We use this histogram (and the corresponding plots for mean, stdev, and stdevVar) to check that, overall, the data in the compared run is not far from the data in the reference run. If, for any variable, either the rms or the absolute value of the distribution mean exceed 1.5, a warning message is issued and stored in the text file. Above 3, a "General Silicon Alarm" goes off.

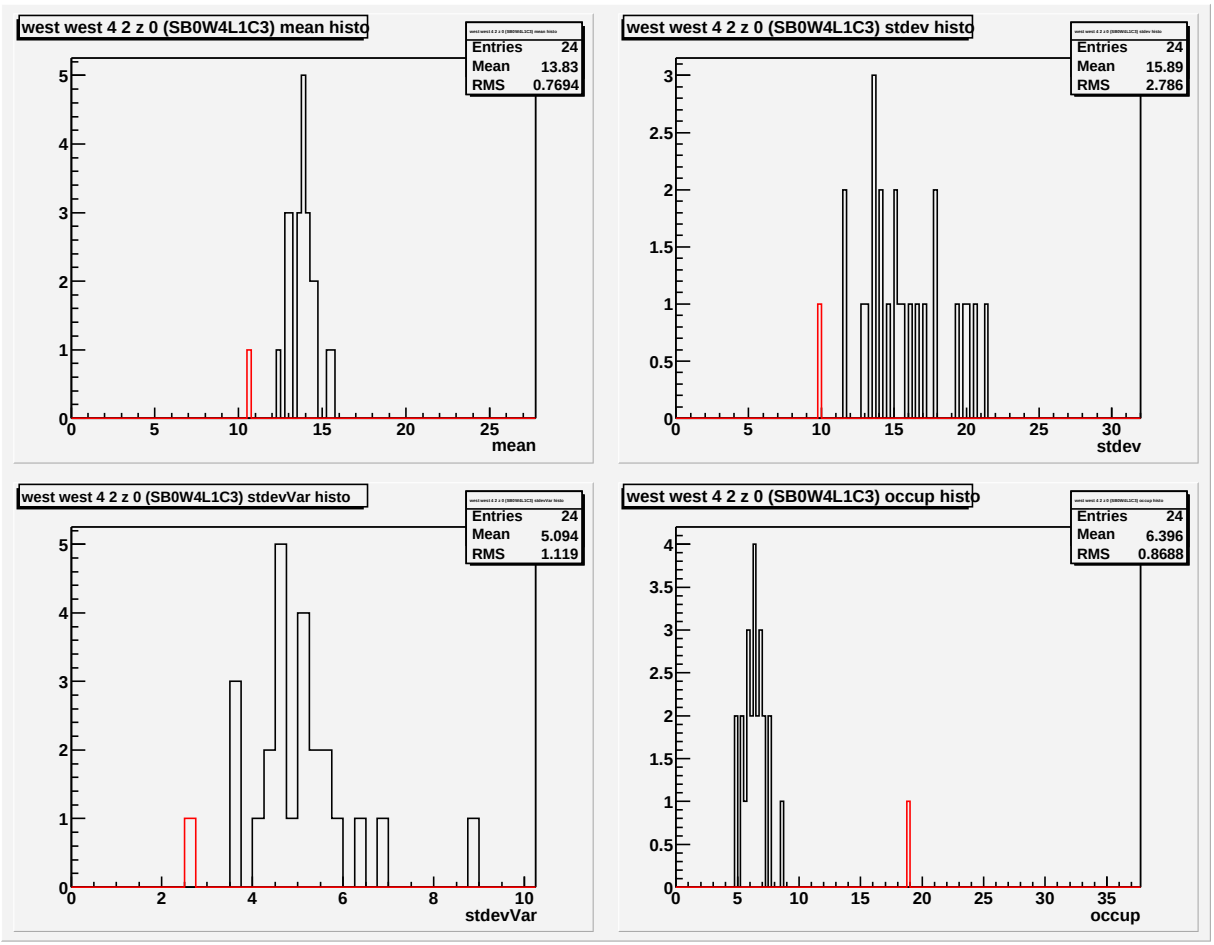


Figure 29: Single chip occupancy, mean charge, charge standard deviation, and stdevVar, for reference runs (black line) and the run being compared (red line). The data for this chip looks like its ladder was unbiased, and an alarm should go off.

- removeSvxMonRun

  Usage:

  ```
  > removeSvxMonRun runNumber TreeFile
  ```

  `removeSvxMonRun` removes the run "runNumber" from the ntuple stored in the file "TreeFile".

- modifySvxMonRun

  Usage:

  ```
  > modifySvxMonRun runNumber TreeFile variable newvalue passwd
  ```

  `modifySvxMonRun` modifies the value of the variable "variable" in the entry "runNumber" of the ntuple. This executable is used to tag a run as good or bad, and "isAGoodRun" is currently the only variable which can be modified. The only purpose of the password is to prevent uneducated users from running this executable. The correct passwd value is "1975".

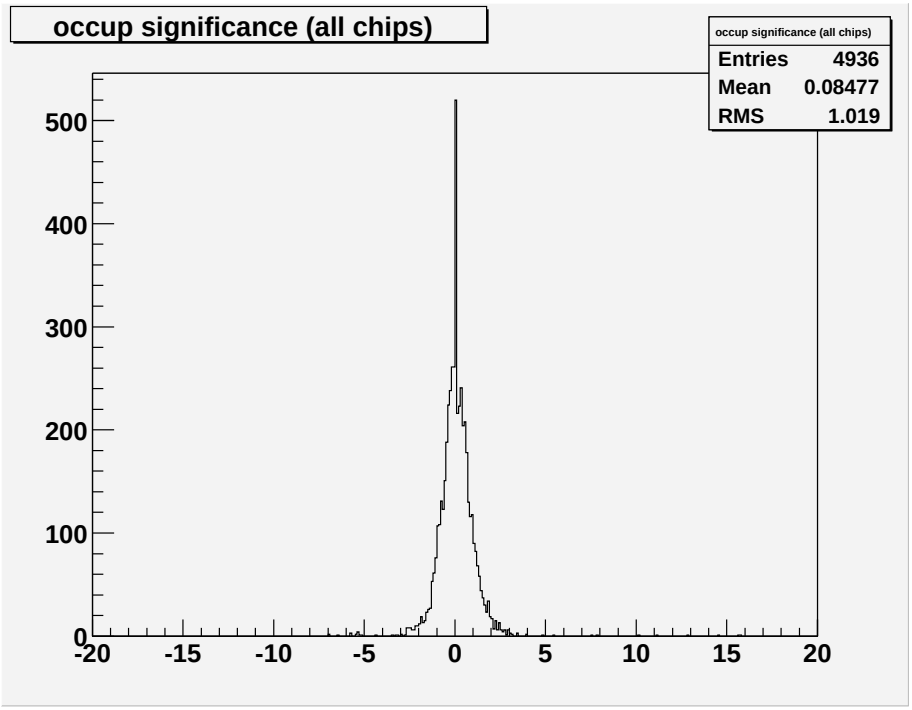


Figure 30: Occupancy significance of all chips in the run being compared. The spike at 0 comes from chips that never send any data (due to truncated readout chains). The rest of the distribution is a Gaussian centered at 0, with an rms close to 1: the run shows no deviation from the reference runs.

6.1.3 Chip History

The SvxMonRunCompare ntuple is a usual ROOT ntuple. As such, it can be accessed either interactively in a ROOT session, or with the help of some ROOT scripts. Example scripts are available on the online machines:

```
~bachacou/SvxMonRunCompareScripts/start_chip_plotter.C
~bachacou/SvxMonRunCompareScripts/start_ladder2D_plotter.C
```

`start_chip_plotter.C` plots the occupancy, the mean charge, the charge standard deviation, and stdevVar of a single chip for a given run (red line) and for a range of reference runs (black line). The reference runs are "good runs" (isAGoodRun != 0) for which SVXMon processed at least 200 events (nSvxMonEvents >= 200). Figure 31 shows such a plot for the mean charge of chip SB0W0L0C0 (first chip on phi side). `start_ladder2D_plotter.C` plots one of the four quantities of all chips on one side of a ladder vs. the run number. Figure 32 shows such a plot for the occupancy of ladder SB0W0L0, phi side. In order to start the scripts, type the following commands on any online machine:

```
source ~cdfsoft/cdf2.cshrc
setup cdfsoft2
root start_ladder2D_plotter.C
```

Then follow the instructions.

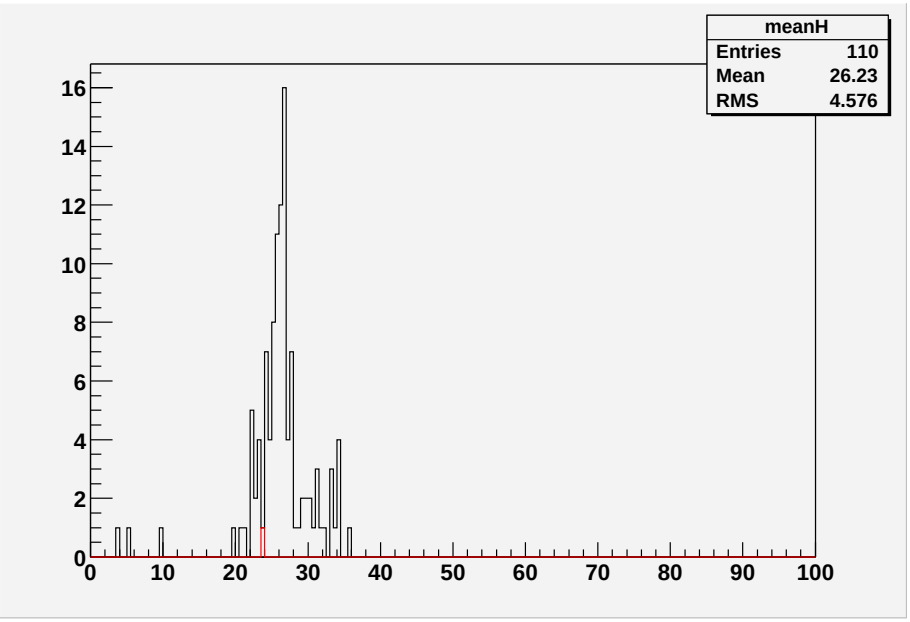


Figure 31: Chip Mean Charge for a given run (red line) and a set of reference runs (black line).

6.2 RootCompare

RootCompare is a small program which helps comparing ROOT histograms from different files. In particular, it can be used to compare SVXMon histogram files from different runs in order to pin down changes from run to run or to compare run plots with reference plots. The program estimates how different the histograms are and allows the user to pay attention only to histograms which change significantly from one run to another.

6.2.1 Starting RootCompare

The script that starts RootCompare is in the directory `~cdfdaq/bin` on B0 machines. It can be started as follows:

```
rootcompare [-p 'pattern'] [-q 'veto-pattern'] file1.root file2.root ...
```

Invoke `rootcompare` without any arguments to see a short usage summary. The optional switch `-p` can be used to load only a subset of histograms from the given files. The histogram names will be matched against the given pattern using shell-like globbing, with characters *?[] acting as usual wildcards. For example,

```
rootcompare -p '*stdev*' file1.root file2.root
```

will select all histograms whose names contain the string "stdev". Other regular expressions like "?" or "[a-z]" can be used as well. See the description of tcl command "string match" for complete details [17]. It is usually prudent to enclose your pattern in single quotes to prevent file name substitutions by the shell.

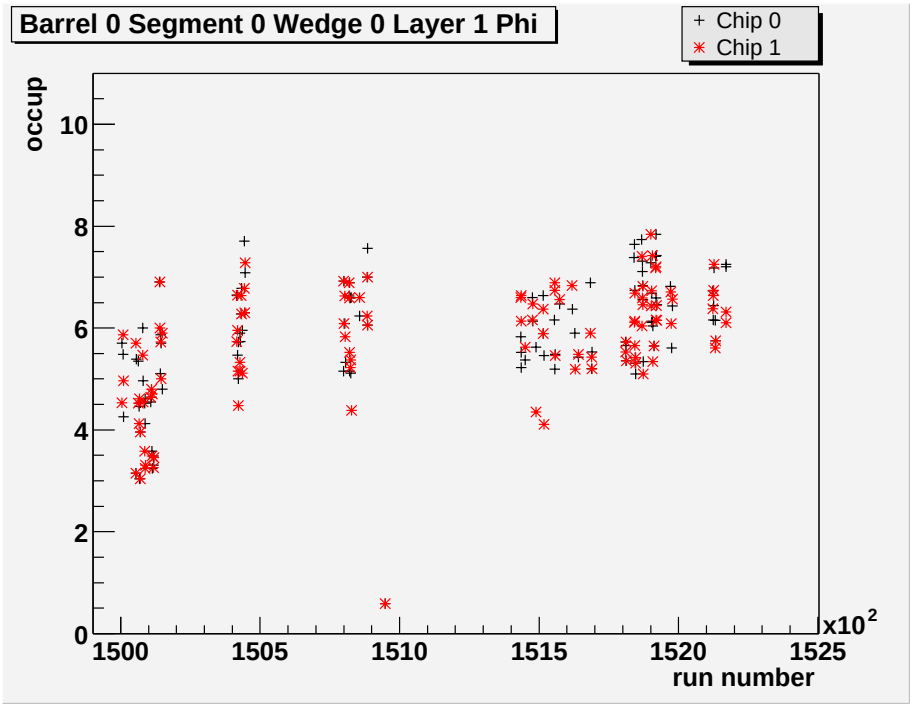


Figure 32: Chip Occupancy vs Run Number.

The optional switch -q works in the same way as -p, and can be used to skip histograms whose names match the pattern. Indeed, loading a large number of histograms is time- and memory-consuming. It is often a good idea to reduce the set of selected histograms. For example, when loading an SVXMon histogram file, it may be useful to specify the options `-q '*strip*' -q '*cell*'` in order to avoid reading many thousands of histograms related to pipeline cell ids and strip-level quantities.

Up to 12 files can be compared at the same time. However, for files containing a large number of histograms, comparing more than three or four becomes too time-consuming because the number of pairwise histogram comparisons increases proportionally to the number of files squared.

The script that starts RootCompare on fcdfsgi2 is ~bachacou/bin/rootcompare. It works in the same way as on B0 machines.

6.2.2 Basic Features

RootCompare finds all histograms in the files provided on the command line (traversing the hierarchy of directories, canvases, and pads) and matches histogram names in different files. When matches are found, dissimilarity coefficients are computed for each pair of histograms with the same name and binning (the dissimilarity coefficients are explained in detail in section 6.2.4). As illustrated in Fig. 33, the RootCompare GUI displays the list of these histograms together with the dissimilarity coefficients. When three or more histograms are compared to each other, the coefficients displayed are the largest ones out of all possible pairwise comparisons.

Figure 33: RootCompare GUI

By default, histograms are sorted by the dissimilarity coefficient D_{rel} in decreasing order. The sorting can be changed by clicking on the coefficient name at the top of the list. The plots can also be sorted alphabetically by name.

Double-clicking on a histogram name in the list produces a ROOT canvas containing one or more histograms, depending on the options.

6.2.3 Display Options

The top part of the RootCompare GUI contains several option-setting elements whose functions are explained below.

- The three buttons "Combo", "Diff", and "$<|\mathrm{Diff}|>$" determine what histograms will be displayed.

  "Combo" : histograms with same names from the different files are plotted on top of each other (for 2d histograms, next to each other)

  "Diff" : only available when comparing two files. Displays the bin by bin difference of the two matched histograms.

"< |Diff| >" : display the average absolute difference between the histograms.

- When the option "New canvas" is selected, the canvases previously plotted remain intact and a new canvas is created when the user double-clicks on a histogram name.
- "1d" and "2d" radiobuttons switch between the lists of one-dimensional and two-dimensional histograms.
- "2d lego" option specifies that two-dimensional histograms should be plotted with the ROOT drawing option "lego2" (using 3d vertical bars) rather than "zcol" (using color cells).
- "Titles match" and "Search" entries accept any pattern with the usual wildcards. "Titles match" option reduces the list of histogram names currently displayed by the GUI to the ones matching the pattern. "Search" highlights one of the histograms in the list matching the pattern. The match is established using the same rules as for the pattern given on the command line after the "-p" switch (emacs afficionados should not forget to add a star to their patterns).

6.2.4 Definition of the Dissimilarity Coefficients

Let $\|Q\|$ denote the norm of a histogram defined as:

$$\|Q\| = \sum_{\mathrm{bin}=0}^{N-1} |\, Q_{\mathrm{bin}} \,|,$$

where N is the number of histogram bins. Then the absolute dissimilarity coefficient D_{abs} between two histograms A and B of the same quantity is defined as

$$D_{\mathrm{abs}} = \frac{\|A - B\|}{N}$$

which is an average absolute difference. For our purposes this coefficient is more useful than chi-square because it puts less emphasis on deviations due to outliers and depends more on the common trend. This coefficient is useful for looking at the plots for which the absolute shifts are important (for example, pedestals).

The relative dissimilarity coefficient D_{rel} is defined as follows:

$$D_{\mathrm{rel}} = \frac{\|A - B\|}{\|A\| + \|B\|}$$

This coefficient is useful for comparison of the quantities which are naturally non-negative, such as occupancy or noise. Its value can not be less than 0 or more than 1.

D_{scale} is the same as D_{rel} but after normalization of the two histograms to the same area. D_{scale} provides the information about the difference of plot shapes when the difference in normalization is not significant. It is useful for histograms whose numbers of entries depend on the number of events processed.

7 SVXMon Configuration

The number of histograms which can be created by SVXMon is virtually unlimited (for example, it can create several types of history plots for any strip in the silicon tracker). Therefore, SVXMon configuration can not be reduced to a simple assignment of some values to a set of predefined parameters. Because of this, the standard AC++ "talk-to" system [10] is not well suited for the job of configuring SVXMon. Instead, the program exposes some of its functionality to tcl by creating a set of tcl commands which can be used to book various plots, set and examine program parameters, and to perform other operations useful for silicon monitoring.

SVXMon configuration may be altered and new plots may be created both in its configuration file and in the middle of event processing. At the beginning of a job the monitor installs a special signal handling procedure[3]. When the user types Ctrl-C into the terminal where SVXMon runs, the program receives an interrupt signal and produces an interactive prompt which can be used to examine the monitor state, make new histograms, save monitoring plots, *etc.* Unlike the AC++ menu shell, SVXMon prompt is a full-featured tcl shell which allows for a complete access to the results of all executed commands (Ref. [11] discusses the AC++ shell shortcomings in more details and suggests a better alternative). At the prompt, the normal program execution can be resumed by typing "quit" or "exit", and typing "stop" will end the job.

7.1 SVXMonModule Parameters

SVXMon utilizes a special AC++ module called SVXMonModule. Although the configuration of this module is still performed using a "talk-to", the exact mechanism is a bit unusual: most of the configuration is performed using a single parameter called "Config_Script". From the point of view of the "talk-to" system, SVXMonModule has the following options:

- **General.** Invokes the "General" submenu. The parameters in this submenu are

  Debug_Print Boolean parameter which turns on or off various debugging messages. The default value of this parameter is *false*.

  Save_MemoryMap_File Boolean parameter which determines whether the silicon monitor will create a memory-mapped file in order to communicate plots and histograms to a display program. The default value of this parameter is *false* (socket communication seems to be more reliable, even on the same machine).

  MapFileName String parameter which specifies the name of the memory-mapped file. It is used only when **Save_MemoryMap_File** is true. The default value of this parameter is "*SVXMon.map*".

  SharedMemorySize Integer parameter which determines the size of shared memory used for root objects (histograms and plots), in kbytes. This parameter is significant only when **Save_MemoryMap_File** is true. Unfortunately, in ROOT it is virtually impossible to determine how much space is needed for various objects until these objects are actually written out. You'll have to experiment if you

[3]This feature may be disabled by setting the SVXMonModule parameter "SmartPrompt" to 0.

need to create many plots simultaneously. The program may experience random crashes if the requested shared memory size is too small. The default value of *15000* is probably not sufficient for most purposes.

Save_ROOT_File Boolean parameter which determines whether the silicon monitor will save its plots in a .root file at the end of job. The default value of this parameter is *true*.

RootFilePrefix String parameter which specifies the prefix for the .root file name. If **Save_ROOT_File** is true then at the end of job monitoring plots will be saved in a file named "RootFilePrefix_XXXX.root" where XXXX is the run number. The default value of this parameter is "*svxmon*".

Start_Server Boolean parameter which determines whether the silicon monitor should start the Server program. Although *false* is the default value of this parameter, it should be set *true* for normal online monitoring.

Update_Plots Boolean parameter which, when set true, forces monitoring plot updates even if the monitor is not connected to the display program. This may be useful for creation of history graphs. *false* is the default value of this parameter.

EnableHRR Boolean parameter which enables or disables Halt-Recover-Run requests to run control. Such requests can be issued by SVXMon when a sufficiently large fraction of silicon readout chips loses pipeline synchronization. The default value of this parameter is *false*.

UseConsumerErrorSender Boolean parameter which determines whether the silicon monitor will use the consumer framework error sender/receiver scheme in order to pass error messages to the central CDF online error logging facility Merlin. The default value of this parameter is *false*.

ErrorReceiverHost String parameter which specifies the name of the machine which runs the consumer framework error receiver program. This parameter is used only when **EnableHRR** or **UseConsumerErrorSender** parameter is true. The default value is "*b0dap63.fnal.gov*".

ErrorReceiverPort Integer parameter which specifies the port number used by the consumer framework error receiver program. This parameter is relevant only when **EnableHRR** or **UseConsumerErrorSender** is true. The default value is *9040*.

Start_Error_Logger Boolean parameter which determines whether the silicon monitor will start a child error logger program, normally the SVX Error Logger. SVXMon will communicate with this program via TCP/IP sockets. The default value of this parameter is *false*.

Error_Logger_Exe This string parameter specifies the name of the child error logger program with full path. The default value of this parameter is an empty string. Therefore, this parameter must be specified by the user whenever the **Start_Error_Logger** value is true. To use the SVX Error Logger, set this parameter to "$env(CDFSOFT2_DIR)/SvxDaqMods/test/ErrorLogger/start_logger". Note that SVX Error Logger is a tcl script which must be used with Tcl/Tk

version 8.3 or newer. It also requires tcl extensions called "BLT" [5] and "Signal" [6]. Because of that, it will fail to start on the machines which lack proper Tcl/Tk/BLT/Signal installation.

Error_Logger_Port Port number for the child error logger. It is used only when **Start_Error_Logger** is true. The default value of this parameter is *9101*.

Error_Logger_Bufsize Number of error messages to buffer for sending to the child error logger. It is used only when **Start_Error_Logger** is true. The default value of this parameter is *1*. In case the frequency of errors in the data stream is high, this parameter should be set to a larger number (100 or so). No matter what the buffer size is, the monitor will send accumulated error messages to the logger at least as often as it updates its histograms.

Error_Summary_File Name of the text summary file produced by the error logger at the end of a run. The default value of this parameter is an empty string which means that the file name will be selected automatically for every run. This parameter is meaningful only when **Start_Error_Logger** is true.

CheckIfIntegrated Boolean parameter which, if true, instructs SVXMon to determine whether the silicon tracker was included in each run encountered in the data stream by checking the USESRC column in the run configuration database. Although the default value of this parameter is *false*, it is important to set it "true" for normal online monitoring.

MaxPassPerRun Integer parameter which specifies the largest number of events to pass in any single run for subsequent writing to disk by one of the AC++ output modules. The default value of this parameter is *0*.

MaxPassTotal Integer parameter which specifies the largest number of events to pass for subsequent writing to disk in one job. The default value of this parameter is *0*.

SmartPrompt Boolean parameter which determines if SVXMon should produce its own prompt when the data processing is interrupted by SIGINT. The default value of this parameter is *true*.

- **SIXD.** Invokes the "SIXD" submenu used to configure the following parameters

  SIXDMonitor Boolean parameter which turns silicon monitoring on or off. It should be left at its default value, *true*.

  Config_Script String parameter. This is the main configuration script for the silicon monitor which is evaluated when SVXMon encounters the first "begin run" record. The default script is "*puts {WARNING : empty SIXD Monitor configuration script}*".

The tcl interpreter used to evaluate the script "Config_Script" is a dedicated interpreter which understands SVXMon-specific commands called `SIXDMon`, `histo`, and `svx`. The fine tuning of the silicon monitor functionality as well as the actual creation of all monitoring plots happen when the script is executed. All standard tcl commands may be used in the configuration script as well.

7.2 Tcl Command "SIXDMon"

The special tcl command `SIXDMon` can be used to set or examine SVXMon internal parameters, either from the configuration code in "Config_Script" or from the special SVXMon prompt when the program is running. Usage is as follows:

```
SIXDMon option arg ?arg ...?
```

The following options may be specified:

SIXDMon configure *paramName value*
: Sets the value of *paramName* to *value* if *paramName* is a valid parameter name and *value* is an acceptable definition for this parameter. This command returns an empty string.

SIXDMon cget *paramName*
: Returns the value of parameter *paramName* if *paramName* is a valid parameter name.

SIXDMon parameters
: Returns the list of valid parameter names.

SIXDMon paramtable
: Prints parameter values to the standard output in a tabular form.

SIXDMon help *topic*
: Prints a help message on a given topic. Type "SIXDMon help" to see the list of help topics.

SIXDMon info *paramName*
: Prints the description of *paramName* or just the type of the parameter if no description is available.

SIXDMon dumplimit *error_type ?detector_element? limit*
: Sets the limit on the number of events dumped to disk due to various readout problems. The limits are applied per error type for each detector element. When the *detector_element* argument is omitted, the command sets the default limit for the given error type on all detector elements. The error types and the names of the detector elements can be seen from the error messages printed to the standard error. *limit* must be an integer. The event dumps for a given error and detector element are suppressed if limit is set to 0 or to some negative value.

7.3 SIXDMon Parameters

Valid SVXMon parameter names which can be set or examined with the `SIXDMon` command are listed below. The numbers or strings shown inside parentheses to the right of the parameter names represent the default values. Some parameters are used only at the beginning of the job, so changing them in the middle of a run will have no effect on the program behavior. The parameters which do alter the program behavior when changed in the middle of the run are marked with asterisk (*) in front of the parameter name.

* **useFullDetector** (1)
* **layerMin** (0)
* **layerMax** (10)
* **layerInRange** (1)
* **zSegmentMin** (0)
* **zSegmentMax** (10)
* **zSegmentInRange** (1)
* **phiWedgeMin** (0)
* **phiWedgeMax** (100)
* **phiWedgeInRange** (1)
* **usePhiSide** (1)
* **useZSide** (1)

  All these parameters collectively specify the silicon system subset for which you want to collect statistics and make plots. By default, the statistics are collected for all ladders encountered in the data stream at least once. If this is not the desired behavior then the boolean parameter **useFullDetector** should be set to 0. The desired layers can be specified using integer parameters **layerMin**, **layerMax**, and boolean parameter **layerInRange**. For example, to collect statistics only for SVX, set **layerMin** to 1, **layerMax** to 5, and **layerInRange** to 1. Setting **layerInRange** to 0 with the same values of **layerMin** and **layerMax** will instead select L00 and ISL. The parameters which specify z segments and wedge numbers work the same way. Boolean parameters **usePhiSide** and **useZSide** can be used to discard the data for the phi and z side of all ladders. For example, if the data are needed for the phi sensor side only, set **useZSide** parameter to 0.

badBunchCrossings ({})

Tcl list of integers which specifies bunch crossing numbers (as defined by the Silicon Readout Controller, SRC) ignored by SVXMon. This parameter may be used to discard bunch crossings in which the Level 1 trigger was issued for a pipeline cell which stored its data when the preamp reset signal was active. Such bunch crossings used to be a common problem until SRC was completely debugged.

* **calibDataVersion** (-1)

  Integer. SISTRIPDH table version for the calibration database access. Negative value means use the default table provided by CalibrationManager.

* **calibRunNumber** (-1)

  Integer. SISTRIPDH table run number for the calibration database access. Negative value means use the default table provided by CalibrationManager when reading, current run number when writing.

* **calibStatus** ("COMPLETE")

  String. SISTRIPDH table status for the calibration database access.

* **checkLadderSet** (1)

  Boolean. Set to 1 in order to check at the beginning of each run that the set of ladders included in the run did not change since the SVXMon start time. It may be useful to turn this check off if there are some problems with database access.

*** checkVrbErrors** (0)
Boolean. Set to 1 in order to check the silicon unpacker status words for problems and to produce corresponding error messages.

*** chipsNeededForHRR** (50)
Integer. The minimal number of silicon readout chips with desynchronized pipeline needed to trigger the Halt-Recover-Run requests to run control. The requests will be issued only if the SVXMonModule parameter "*EnableHRR*" (section 7.1) is set true and the pipeline status map is created using the "*histo cellid map*" configuration command (section 4.8).

*** databaseTag** ("auto")
String. Determines the database which will be used to fetch the silicon configuration for each run (the set of ladders, chip DPS modes, *etc*). The allowed values are "auto", "onotl_prd_read", and "ofotl_prd_read". "auto" means that the online database will be chosen on B0 machines and the offline database will be used in all other cases.

*** debugFlag** (value of SVXMonModule parameter "*Debug_Print*", section 7.1)
Boolean. Set to 1 in order to print various debugging messages, set to 0 to suppress them. This parameter is useful for turning debugging messages on or off in the middle of a run.

*** defaultDumpLimit** (2)
The number of events to dump when the program encounters various errors in the data stream. The dump limits are defined per error type for each detector element. This parameter sets the global default limit which is used to initialize error counters when there are no other dump limits defined. See also the description of "*SIXDMon dumplimit*" command (section 7.2) for setting limits with better granularity.

*** destructAll** (1)
Boolean. Set to 0 in order to skip the normal object destruction sequence when the program exits. In this way, the use of inefficient ROOT garbage collection mechanism is avoided, and the program completes significantly faster when it has a large number of histograms.

*** detailedVrbErrors** (0)
Boolean. Set to 1 in order to print detailed info about every mismatch in bunch crossings, trigger supervisor counter values, and time since level 1 accept.

discardBadStrips (0)
Boolean. Set to 1 in order to discard bad strips when the program calculates various statistics for chips and ladders. Used only when **tagBadStrips** parameter is also set to 1.

*** endJobScript** ({})
String. This is a tcl script which is executed at the end of job. A good place to prints some statistics, write list of bad channels into the database, *etc*.

* **historyFrequencyDivider** (1)

  Integer. Together with the set of parameters which define the snapshot schedule, this parameter specifies how often SVXMon updates history records created by the "*svx watch*" command (section 7.4). The records are made just before a new snapshot is created, so that the histories of snapshot plots are based on the largest number of events possible. Setting the **historyFrequencyDivider** to a positive number N will trigger one history record update for every N snapshot changes. Setting this parameter to 0 or to a negative number will disable the history recording mechanism.

hitsNtuple (0)

  Boolean. Set to 1 in order to create an n-tuple of silicon tracker data. This n-tuple is an ultimate memory hog, and can only be used with small datasets for debugging and testing purposes.

* **hrrIntervalSec** (300)

  Integer. The minimal time interval in seconds between two Halt-Recover-Run requests.

monitorVrbOccupancy (0)

  Boolean. Set to 1 in order to monitor VRB-level silicon tracker occupancy. This occupancy is determined in the crates using events which pass level 1 trigger, with much higher statistics than in the normal data stream. The data are relayed from the crates with a help of a separate program (vrb occupancy transmitter) which must be started before SVXMon.

* **neventsHardUpdate** (1000)
* **neventsSoftUpdate** (100)
* **timeHardUpdate** (60)
* **timeSoftUpdate** (3)

  This set of integer parameters defines how often SVXMon updates its plots. The plots are not updated after each event (the socket communications and the display program are just too slow). Instead, the monitor will decide when to redraw histograms using a simple scheduling algorithm. The histograms are updated if at least one of the following three statements is true:

  1) The number of events processed since the last update is equal to or greater than **neventsHardUpdate**.

  2) The time elapsed since the last update is equal to or greater than **timeHardUpdate** seconds.

  3) The number of events processed since the last update is equal to or greater than **neventsSoftUpdate** and, simultaneously, the time elapsed since the last update is equal to or greater than **timeSoftUpdate** seconds.

* **neventsErrorAlert** (5)

  Integer. SVXMon increases the severity level of certain types of error messages when identical errors are encountered in **neventsErrorAlert** or more consecutive events. For example, if a wrong pipeline cell number is found on one of the chips, it could be due to a bit transmission error over the optical cable or due to a glitch on the front-end

clock line. The former problem may be sporadic, while the latter will manifest itself in many consecutive events, and should be treated more seriously by the shift crew.

*** periodicScript** ({})
String. This is a tcl script which can be executed periodically by SVXMon. The frequency of execution is set by the **scriptPeriod** parameter.

*** scriptPeriod** (0)
Integer. This parameter determines how often the program executes the tcl script defined by the **periodicScript** parameter. If **scriptPeriod** is positive then **periodicScript** is executed every **scriptPeriod** events. 0 or negative values of this parameter disable the periodic script.

*** pulseThreshold** (−300)
Integer. Silicon monitor ignores all hits with pulse heights less than **pulseThreshold**.

*** reportingMode** (0)
Integer. This parameter affects the way SVXMon reports certain types of silicon bank unpacker errors. The unpacker may raise several error flags in the HDI status word. When reporting mode is 0, SVXMon makes a separate message for every flag. In mode 1 SVXMon tries to report only the problems which are likely to be caused by independent failure mechanisms. All other **reportingMode** values are equivalent to 0.

scanPeriod (0)
Integer. Used to determine the number of events in each scan cycle for all plots produced by the "*histo scan*" commands (section 4.10).

snapshotStrategy (0)
Integer. All currently implemented silicon monitoring plots support the concept of snapshots. When a snapshot is made, a copy of the relevant data structure is created which is not affected by data accumulation anymore. This copy usually overwrites the oldest existing copy of the same data structure. The short-term (snapshot) plots display the *difference* between the current value of associated data accumulation object and its oldest snapshot.

Most monitoring histograms are filled using the data stored in the SiStripTimedSet object (this object accumulates various statistics for every strip in the silicon system). The value of **snapshotStrategy** defines how many different copies of the SiStripTimedSet object we are going to keep: **snapshotStrategy** less than or equal to 0 means no snapshots, 1 means one snapshot, 2 or more means two snapshots. Two snapshots result in a smoother update of all short-term plots derived from SiStripTimedSet but require more memory than one snapshot.

Note that each 2-d histogram (created with the "*histo strips2d*" command, section 4.1) has its own associated data structure and its own value of snapshot strategy parameter.

*** snapEventsHardUpdate** (0)
*** snapEventsSoftUpdate** (0)

*** snapTimeHardUpdate** (0)
*** snapTimeSoftUpdate** (0)

This set of integer parameters defines the frequency of snapshot updates. The snapshot updating algorithm is the same as the algorithm for updating monitoring plots (see the description of **neventsHardUpdate** parameter earlier in this section). Note that by default the silicon monitor will try to update histograms and to make snapshots only when it thinks that it may be connected to a display program. You can force updates by changing the value of SVXMonModule parameter "*Update_Plots*" (section 7.1) to "true".

*** storeHistosOnly** (0)

Boolean. Set to 1 in order to store only monitoring histograms and n-tuples in a .root file. The plot layout structure provided by canvases and pads will be discarded.

stripsNtuple (0)

Boolean. Set to 1 in order to create an n-tuple of silicon strip data at the end of job. For each strip, the following set of variables is saved in the n-tuple:

barrel : barrel number,
segment : ladder segment,
layer : detector layer,
phiwedge : phi wedge,
side : sensor side (0 for phi and 1 for z),
strip : strip number on the sensor,
discard : set to 1 if the strip has been discarded in calculations of various statistics such as detector occupancy,
events : number of readouts for this strip,
mean : average pulse height,
stdev : standard deviation of the pulse heights,
skew : pulse height distribution skewness,
kurt : pulse height distribution kurtosis.

stripsNtupleHasDnoise (0)

Boolean. If set to 1 then the n-tuple of silicon strip data also includes **dnoise** as its last variable. Dnoise is the standard deviation of the pulse height difference between adjacent channels divided by $\sqrt{2}$. It makes sense to set this parameter to 1 only when the silicon system is in "read all" mode.

tagBadStrips (0)

Boolean. Set to 1 in order to retrieve the list of bad strips from the calibration database at the beginning of a job.

*** unpackerMonMode** (0)

Boolean. Set to 1 in order to use the monitoring mode of the "nearest neighbor" silicon bank unpacker. This parameter is used only when the **useNNUnpacker** parameter is set to 1.

*** useNNUnpacker** (1)
Boolean. 0 means use the "classic" silicon bank unpacker, 1 means use the "nearest neighbor" unpacker.

vrbOccupancyHost ("b0dap30.fnal.gov")
String. Specifies the name of the machine which runs the vrb occupancy transmitter program. This parameter is used only when the value of **monitorVrbOccupancy** parameter is "true".

vrbOccupancyPort (9301)
Integer. Specifies the port number on which the vrb occupancy transmitter program accepts connections. This parameter is used only when the value of **monitorVrbOccupancy** parameter is "true".

vrbOccupancySnapshots (0)
Boolean. Set to 1 in order to allow updates of VRB occupancy histograms accumulated over short periods of time. This parameter is used only when the value of **monitorVrbOccupancy** parameter is "true".

7.4 Tcl Command "svx"

The special tcl command **svx** is used to perform various SVXMon configuration operations besides creation of plots and parameter manipulation. Usage is as follows:

```
svx option arg ?arg ...?
```

Valid options are:

svx rootfile *fileName*
Saves all monitoring plots in a ROOT file with given name.

svx numchips *detector*
Returns the number of chips on the detector specified. *detector* specifies one face of a silicon half-ladder in the standard SVXMon format [8].

svx det2name *detector*
Returns a short name for the detector specified. This detector name is used inside names and titles of various histograms and canvases created by SVXMon. The naming convention follows the scheme originally proposed in Ref. [15]. It is illustrated in Fig. 34.

svx name2det *detectorName*
This is the inverse of the **svx det2name** command. Returns the detector specifier in the standard SVXMon format [8] for the sensor with the given short name.

svx detectors
Returns the list of silicon half-ladder sides "known" to SVXMon at that point. Each detector specification is itself a tcl list [8]. A detector becomes "known" to the program when it is encountered in the data stream, when the list of bad strips for this detector is loaded from the calibration database, or when bad strips for this detector are defined by the **svx badstrips** command.

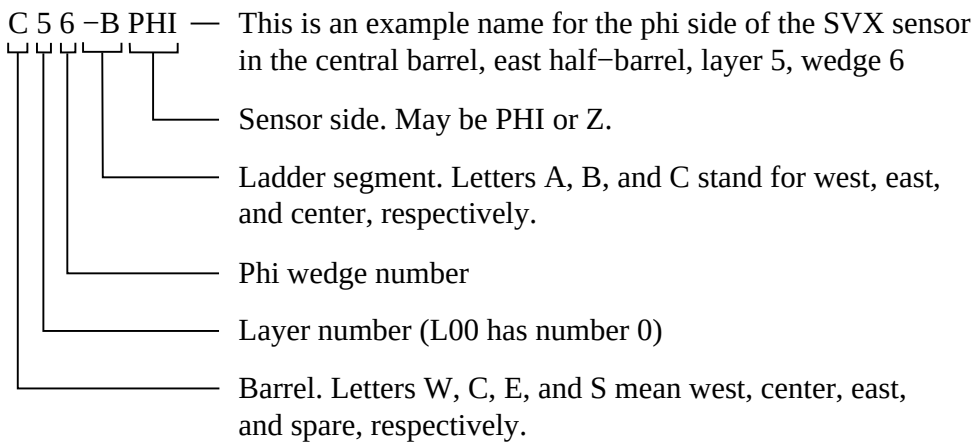


Figure 34: Sensor naming convention used in the titles of various SVXMon histograms and canvases.

svx numerology *option*

Returns the list of half-ladders in the silicon system. Each element of this list is a standard SVXMon detector specifier [8]. The following *option* values are allowed:

N any positive integer is treated as a CDF run number. The list of ladders included in the given run is retrieved from the run conditions database. The command throws a tcl error and prints a few diagnostic messages to the standard error if the database doesn't have the silicon system info for this run.

"all" return the list of sensors in L00, ISL, and SVX (but not SOC)

"kitchen_sink" all sensors in the system, including SOC

"l00" return L00 sensors only

"svx" SVX sensors only

"isl" ISL sensors only

"soc" SOC sensors only

svx runtype *runNumber*

Returns the run type for the given run number as an integer (see [12] for the list of CDF run types).

svx badstrips *operation arg? arg...?*

Tags and untags bad strips. When a strip is tagged as bad, it may be removed from various chip and ladder averages, occupancy calculations, *etc.* The following operations are supported:

svx badstrips get *detector* — returns the list of bad strips for the specified detector [8].

svx badstrips set *detector stripList* — sets the bad strips for the specified detector. *stripList* is a tcl list of integers.

svx badstrips unset *detector* — clears the bad strip tags for the specified detector. If the *detector* argument is omitted then this command removes bad strip tags from all strips in the silicon system.

svx badstrips count *detector* — returns the number of bad strips for the specified detector. If the *detector* argument is omitted then this command returns the number of bad strips in the whole silicon system.

svx badstrips tag — tries to figure out which strips are bad using the accumulated strip statistics. Returns the number of bad strips on success or -1 on failure. At the time of this writing, the algorithm for tagging individual strips using SVXMon data has not been developed yet.

svx badstrips untag *detector* — this command is identical to **svx badstrips unset** *detector*.

svx badstrips retrieve — fetches the table of bad strips (called SISTRIPDH) from the calibration database.

svx badstrips store — writes the table of bad strips to the calibration database, provided that the user has proper database access permissions. Both this command and the previous one use SIXDMon parameters "*calibRunNumber*", "*calibDataVersion*", and "*calibStatus*" (section 7.3) to determine the corresponding table attributes.

svx badstrips discard — after execution of this command the silicon monitor will discard bad strips in its calculations of occupancies, pulse height averages, *etc.* Only the strips tagged bad when this command is executed will be discarded (these strips receive the "discard" tag). Subsequent changes in the strip good/bad classification do not affect the list of strips to be discarded until the **svx badstrips (un)discard** command is used again.

svx badstrips undiscard — clears all "discard" tags. SVXMon stops discarding bad strips in its calculations of various monitoring quantities.

svx data *operation fileName*

Reads and writes strip statistics data to/from disk.

svx data write *fileName* — writes strip statistics data to an ASCII file. If fileName is specified as "stdout" or "stderr" then the data are written to the standard output or standard error, respectively.

svx data read *fileName* — reads strip statistics data from a file. The statistics accumulated prior to that point are discarded.

svx data append *fileName* — reads strip statistics data from a file and adds them to the statistics already accumulated. This command may be used to speed up processing of big data files by running several SVXMon programs on different event subsets and then combining the results.

svx striptuple

Creates an n-tuple of silicon strip data if it doesn't exist yet. See the description of SIXDMon parameters "*stripsNtuple*" and "*stripsNtupleHasDnoise*" in section 7.3 for the definition of n-tuple variables.

svx setrun *runNumber*

Sets the current run number for the silicon monitoring plots. This command is only useful for debugging since the run number is set automatically whenever a "begin run" record is encountered in the data stream.

svx getrun
Returns the current run number

svx nevents
Returns the number of events processed by SVXMon.

svx getpid
Returns SVXMon UNIX process id.

svx fib2chip *fib_id0 fib_id1 chipNumDown*
Returns the tcl list {*detector chipNumSensor*} for the specified FIB ID0, FIB ID1, and the chip number in the bit stream download sequence. FIB ID1 must include the FIB channel number in its five least significant bits in the manner described in Ref. [14]. The chip number in the sequence, *chipNumDown*, follows the convention adopted by the SiChipKey class: the last chip read out has the number 0. *detector* is the standard SVXMon detector specifier [8]. The returned chip number on the sensor face, *chipNumSensor*, corresponds to the offline strip numbering: the chip with number 0 is the chip which reads out strips 0-127, chip 1 reads out strips 128-255, *etc.*

svx chip2fib *detector chipNumSensor*
Returns the tcl list {*fib_id0 fib_id1 chipNumDown*} for the specified detector and the chip number on the sensor face. This is the inverse of the **svx fib2chip** command.

svx chip2daq *detector chipNumSensor*
Returns the name of the specified chip in the DAQ terminology. For example, "`svx chip2daq {west west 1 3 phi} 2`" returns "SB0W1L2C2". Please see Ref. [16] for a discussion of various silicon naming conventions.

svx fib2daq *fib_id0 fib_id1 chipNumDown*
Returns the name of the specified chip in the DAQ terminology.

svx vrb2chip *detectorType vrbId pcId hdiId chipId*
Returns the tcl list {*detector chipNumSensor*} for the specified hardware ids. *detectorType* is a string which must be one of "svx", "isl", or "l00". *vrbId, pcId, hdiId,* and *chipId* are integers which specify, respectively, VRB id, port card id, HDI id, and chip id in the offline nomenclature.

svx hdichips *fib_id0 fib_id1*
Returns the number of chips connected to the specified HDI.

svx dpsmode *detector chipNumSensor runNumber*
Returns 1 if the specified chip was configured to use DPS in the given run, 0 otherwise.

svx readmode *detector chipNumSensor runNumber*
Returns the readout mode for the given chip in the given run. 0 means read-all mode, 1 stands for sparse readout with neighbors, and 2 for normal sparse readout.

svx canvases *?matchPattern? ?excludePattern?*
Returns ROOT names of the monitoring canvases in a tcl list. If the *matchPattern* is specified and it is not an empty string, only the canvas names which match this

pattern are returned. The match is determined using the rules of the tcl "string match" command. If, in addition, *excludePattern* is specified and it is not an empty string then only the canvas names which do not match this pattern are returned.

svx histos *?classPattern? ?matchPattern? ?excludePattern?*
Returns ROOT names of the monitoring graphs, histograms, and n-tuples in a tcl list. *classPattern*, *matchPattern*, and *excludePattern* are optional strings which specify the subset of names to return. *classPattern* is matched against the ROOT class name of the object. The match is determined using the rules of the tcl "string match" command. For example, pattern "TH2*" will match all 2-d histograms. *matchPattern* is matched against the ROOT object name. Only the names which match this pattern will be returned. *excludePattern* is also matched against the name, but the name will be suppressed in the output if the match is found. If any of these patterns is specified as an empty string, the pattern will have no effect (so that an empty string can be used as a placeholder).

svx folder *?folderName? ?matchPatternList?*
Depending on the arguments provided, this command can be used in three different ways:

svx folder *folderName matchPatternList* — in this form the command specifies a folder into which the monitoring plots will be placed by the consumer display program and a list of patterns which will be used to match the names of ROOT objects which will be placed into this folder. The folder will be created if it does not already exist. If the folder already exists, the new patterns will be appended to the existing ones. The object name match will be determined using the rules of the tcl "string match" command. When more than one pattern matches from different folders, the plots will be placed into the folder defined earlier in the configuration file. In this form the command returns an empty string. Example:

```
svx folder "SVXMon/Layer Plots" [list "Layer *" "Short-term layer *"]
```

The folder name "Slides" is special for the display program. It will perform a slide show using plots placed into this folder.

svx folder *folderName* — in this form the command returns the list of patterns defined for the given folder or an empty list if the folder does not exist.

svx folder — without any additional arguments, the command returns the list of folders defined so far.

All **svx folder** commands should normally precede all plot creation commands in the SVXMon configuration file.

svx classname *objectName*
Returns the ROOT class name of an object named *objectName*. SVXMon keeps a map of object names into object pointers for monitoring canvases, histograms, graphs, and n-tuples, but not for lower-level objects such as pads and labels.

svx canvashistos *canvasName*
Returns ROOT names of the plots placed on the canvas with the given name.

svx numbins *histogramName*

Returns the number of bins for the histogram with the given ROOT name. This command will return a two-element list in case the *histogramName* argument refers to a 2-d histogram. In such a list, the number of bins along the X axis goes first, and the number of bins along the Y axis is the second.

svx getbin *histogramName x_bin_number ?y_bin_number?*

Returns the bin value for the histogram with the given ROOT name. The *y_bin_number* argument must be present for 2-d histograms and it must be absent for 1-d histograms. The bin numbering starts with 0 for visible bins which is different from ROOT convention of using bin number 0 for underflows.

svx watch *histogramName recordType ?infoText? ?x_bin_number? ?y_bin_number?*

Monitors and records the time history of a histogram with the given ROOT name. Please see the description of SIXDMon parameter historyFrequencyDivider (section 7.3) for details about history timing. The *recordType* argument specifies what, precisely, is to be recorded. The following *recordType* values are allowed:

bin or **singlebin** — monitor a single bin. This option requires that the integer argument *x_bin_number* is specified. Both *x_bin_number* and *y_bin_number* must be specified in case *histogramName* refers to a 2-d histogram.

distmean or **distaverage** — monitor the histogram average.

diststdev or **distrms** — monitor the histogram width.

distmedian — monitor the histogram median.

distrange — monitor the histogram range. The range is defined as the difference between 75th and 25th percentiles times 0.7413011 which is equal to the standard deviation for the Gaussian distribution.

binmean or **binaverage** — monitor the average of the bin values. Note that this is not the same as the histogram average since bin positions are not used.

binstdev or **binrms** — monitor the RMS of the bin values. Note that this is not the histogram width since bin positions are not used.

binmedian — monitor the median of the bin values.

binrange — monitor the range of the bin values. As in the case of **distrange**, the range is normalized to be equal to the standard deviation for the Gaussian distribution.

The optional *infoText* argument may be used to provide a description of the monitored quantity. It will be used as a legend entry for this quantity in the history plots. A simple default description will be generated automatically when *infoText* argument is absent or it is an empty string.

The **svx watch** command returns a history record identifier string for subsequent use with the "*histo history*" command (section 4.9).

svx reset

Resets all monitoring plots and data accumulation objects. This command should not be used without a good reason.

svx unpackerReport

Prints to the standard output the summary of errors accumulated so far by the silicon bank unpacker.

svx datarep *?translationScheme?*

Sets or shows the offline silicon data representation. When invoked without the *translationScheme* argument, this command shows the current silicon ADC data translation table between the DAQ and the offline. When the *translationScheme* argument is specified, this command changes the ADC translation scheme to the requested one. Valid schemes are "Run2ComData", "Run2ComData_3BS", "Run2V1", "Run2V1_3BS", and "Unspecified" (select from data).

8 Instructions for Consumer Operators

Because the Consumer Operator (CO) has a limited amount of time to devote to silicon detector monitoring, it is necessary to provide a precise and short set of SVXMon-related instructions which can be followed in a systematic way. The instructions are updated regularly; they can be found on the web [18], and in the CO binder in the CDF Control Room. For up-to-date information and recent reference plots, the CO should always refer to these instructions. Here follows a description of the tools we ask the CO to use.

The two main tools useful for COs are the cell id status map and the chip status map. An occasional look at the SVX Error Logger messages may also be useful, especially if the CO has some prior SVX experience. The CO is also expected to respond to SvxMonRunCompare alarms when necessary.

The status map plots are listed in the "Slides" directory of the HistoDisplayMain GUI devoted to silicon (the bottom rightmost monitor on the CO panel). The SVX Error Logger is displayed on one of the two rightmost monitors. The status maps should be thoroughly checked

- at the beginning of a store.
- at least twice per shift.

Whenever the silicon is integrated and the run type is "physics", the CO should also have a look at the maps at regular intervals during the shift.

8.1 The Cell Id Status Maps

The maps are displayed in two canvases: one canvas displays the SVX layers (SVX L0 to SVX L4), the other displays the L00 (innermost layer) and the ISL (ISL1 and ISL2, the two outermost layers).

8.1.1 How to Read the Maps

- Each histogram corresponds to a subset of the silicon detector. In each histogram, a bin corresponds to a single chip.
- Left column of histograms: phi side chips, right column: z side chips (small angle stereo or 90 degrees, depending on the layer).

- Within a column: 1 Histogram = 1 Layer (L00 and ISL on one canvas, SVX L0 to SVX L4 on the other).
- Within a histogram: X axis is the z-segment (0 = westmost, 5 = eastmost); Y axis is the wedge number (0 to 11 for L00 and SVX, more for ISL).

8.1.2 Color Codes

- Blue: the chip is masked because of a known problem or it is not included in the run.
- Green: OK.
- Red: Failure. The chip pipeline has lost its synchronization with the rest of the system or the data is corrupted.
- Yellow: no data is present for this chip in the data stream after bank unpacking. This is a failure, as serious as "Red"[4].

8.1.3 Response

If the CO sees chips that are "yellow" (no data) all the time, and over several runs, he should write an entry in the general e-log specifying the chip location (layer, phi or z side, z-segment, and wedge), and page the Silicon General pager.

A few isolated "red" chips are acceptable (some chips lose synchronization more or less randomly).

Sometimes, a whole "DAQ wedge" loses synchronization, so that in several layers on both phi ans z sides all the chips in the same segment and wedge are "red", such as DAQ wedges SB2W10 (SVX segment 2, wedge 10) and SB3W1 (SVX segment 3 wedge 1) in Fig. 23. In this case, reinitializing the chips by executing the Halt-Recover-Run sequence from run control or starting a new run might fix the problem. If this doesn't help, the CO should put an entry in the general e-log and page the Silicon General pager.

8.2 The Chip Status Map

The chip status maps are designed in the same way as the cell id maps. The only difference is a slight change in the meaning of the color codes.

8.2.1 Color Codes

- Blue: the chip is masked because of a known problem or it is not included in the run.
- Green: OK.
- Red: Failure. The data, in terms of strip occupancy and charge distributions, is suspicious.
- Yellow: no data is present for this chip in the data stream after bank unpacking. This is a failure, as serious as "Red"[4].

[4]Except for a few minutes at the beginning of a run, when SVXMon does not have enough events to perform its classification. As soon as you see some green and red codes, yellow is a problem.

8.2.2 Response

If the CO sees chips that are "yellow" (no data) all the time, and over several runs, he should write an entry in the general e-log specifying the chip location (layer, phi or z side, z-segment, and wedge), and page the Silicon General pager.

A few isolated "red" chips are acceptable, especially if they match the ones on the cell id status map. Some chips get out of limit for a short time and then come back to normal, the CO should not worry about those. However, if some chips are always red over several runs, he should write an entry in the general e-log, specifying where they are.

If large areas of the silicon detector are "red" or "yellow" during a physics run with collisions, the CO should page the General Silicon pager.

8.3 SvxMonRunCompare Alarms

At the end of each run, SvxMonRunCompare (see section 6.1) checks whether any component of the detector was unbiased (*i.e.*, the "High Voltage" was not on). If it finds any unbiased ladder, a window pops up on one of the CO monitors with an alarm message as in Fig 35. The CO should read the message carefully and follow the intructions.

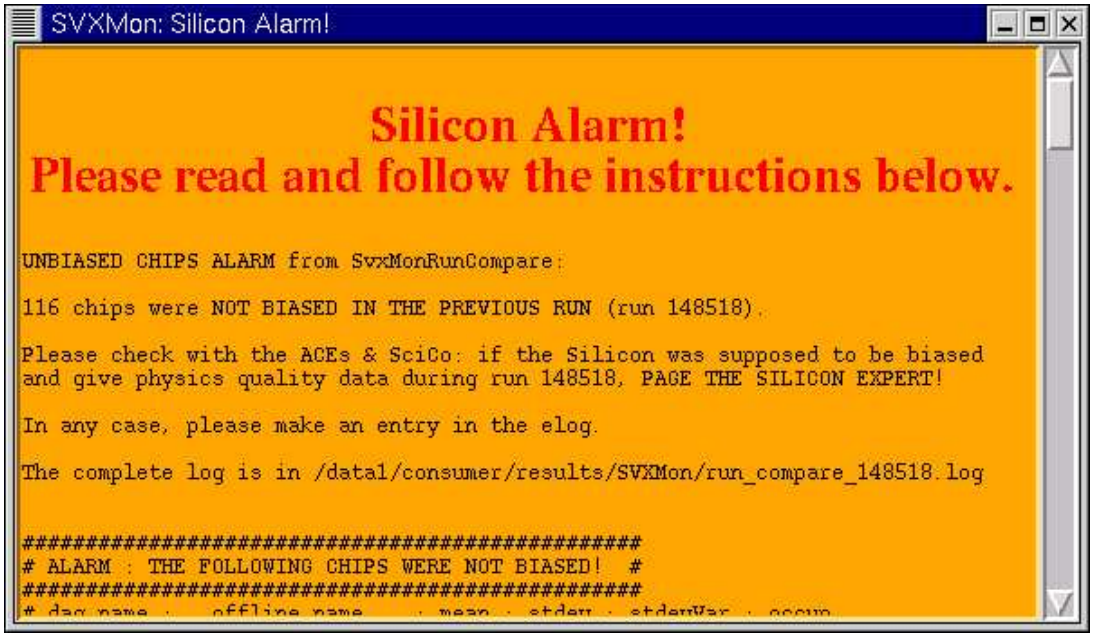


Figure 35: Bias Voltage Alarm Pop-up window

A Compiling SVXMon

To compile SVXMon, one has to log into a computer with Run II CDF software installed and issue the following commands (assuming csh-compatible shell):

```
set release = development   (or your favorite cdfsoft2 release)
source ~cdfsoft/cdf2.cshrc
setup cdfsoft2 $release
cd somedir                  (where you have lots of disk space)
newrel -t $release tmpmon
cd tmpmon
addpkg SvxDaqMods
gmake bin
```

On the CDF online machines such as b0dap32.fnal.gov the above commands will build the silicon monitoring executable bin/$BFARCH/SVXMon. On the offline machines such as fcdfsgi2.fnal.gov one has to run "gmake tbin" instead of "gmake bin". This will create several executables, including SVXMon, in bin/$BFARCH. In order to compile just the SVXMon executable without all the others, one has to edit GNUmakefile in the directory SvxDaqMods/test. Change the "SUBDIRS" definition at the beginning of the file so that it includes only "SVXMon" and comment out the definition of TBINS.

B Compiling SvxMonRunCompare

Compiling SvxMonRunCompare is very similar to compiling SVXMon. Here is the list of commands necessary to compile the four executables:

```
set release = development   (or your favorite cdfsoft2 release)
source ~cdfsoft/cdf2.cshrc
setup cdfsoft2 $release
cd somedir                  (where you have lots of disk space)
newrel -t $release tmpmon
cd tmpmon
addpkg SvxDaqUtils
gmake tbin
```

C Viewing Online Plots at Remote Institutions

When SVXMon (or any other consumer) is running at B0, one has to use ssh and its port forwarding feature in order to view the plots on a computer outside the B0 cluster. Suppose, user desktop machine is called cdfpc1.lbl.gov, and SVXMon is running on b0dap66.fnal.gov serving plots and histograms on port 9091. Then the port forwarding can be set up from cdfpc1.lbl.gov by executing the following command:

```
ssh -g -L 9999:b0dap66.fnal.gov:9091 b0dap30.fnal.gov
```

While this ssh session is in progress, the display program running on cdfpc1.lbl.gov can be connected to port 9999 on the local host. Of course, port number 9999 may be replaced by an arbitrary unused port between 1024 and 65535. The port forwarding can also be set up from b0dap30. This is useful, for example, if you have normal ssh/sshd installed on your machine instead of the kerberized ones:

```
ssh -g -R 9999:b0dap66.fnal.gov:9091 cdfpc1.lbl.gov
```

D SVXMon Messages

SVXMon message types are limited to 20 characters. This restriction is inherited from the ZOOM error logging facility [4] used by SVXMon. The message severity levels are varied according to perceived importance of the detected problem. Severity level 4 ("info") is reserved for summary messages generated at the end of each run.

In the table below, the "Source" column shows where the error diagnostics is generated. In this column, U means that the problem is detected by the unpacker, V means that the error flag is set by VRB hardware, and S means that this particular type of message is generated by SVXMon code. The message types are listed in alphabetical order with the exception of various "Out of Limits" errors. The "Out of Limits" errors are described in detail in section 5.

Message Type	Source	Meaning	Probable Causes
0 Events	S	A plot update was requested before any events were processed	A new run has just begun
Bad ADC Value	S	The ADC value for some channel is invalid (after bank unpacking)	Hardware error in the chip (the ADC value higher than counter modulo + 1) Optical transmission error Bad pedestal subtraction lookup table in the FIB Mismatch in the data representation used by SVXMon and by the bank unpacker
Bad Chip ID	U	Chip number in the data is out of range for a given HDI	Optical transmission error
Bad Configuration	S	SVXMon is unable to configure itself	Online production database is down or unreachable An error in the SVXMon tcl configuration file

Message Type	Source	Meaning	Probable Causes
Bad FIB Id	S	Unrecognized FIB id pair	Ids in the FIB register do not conform to the standard convention
Bad Hit Position	S	Track helix has no intersection with a half-ladder which has a hit on the track	CDF software bug
Bad Parameter Value	S	Invalid parameter value in the SVXMon configuration file	An error in the SVXMon tcl configuration file
Bad VRB Data Format	V	The "Data Format Error" bit is set in the VRB error word of the SIXD/ISLD bank	VRB hardware error
Bit 0 High	U	Bit 0 is stuck high, the unpacker is able to correct the problem	Optical transmission error
Bit 0 Low	U	Bit 0 is stuck low, the unpacker is able to correct the problem	Optical transmission error
Bit 1 High	U	Bit 1 is stuck high, the unpacker is able to correct the problem	Optical transmission error
Bit 1 Low	U	Bit 1 is stuck low, the unpacker is able to correct the problem	Optical transmission error
Bit Error (not 0/1)	U	Stuck bit (not 0 or 1), the unpacker is able to correct the problem	Optical transmission error
Bunch X Out of Sync	S	SIXD/ISLD bunch crossing numbers are different	FIB hardware error Reformatter error
C2 Termination	U	Truncation termination character seen (0xc2)	Silicon hybrid drops readout Optical transmission error
Chip Side Error	U	Wrong chip id in the data stream	Optical transmission error
Configuration Change	S	The set of silicon ladders included in the system has changed while SVXMon was running	The silicon configuration has been modified
Data Found After EOR	U	Data found by the bank unpacker after the EOR record in the HDI block	DAQ error
Data Overrun	U	HDI readout length is 0xffe = 4094 bytes	Silicon hybrid malfunction Optical transmission error

Message Type	Source	Meaning	Probable Causes
DRO_00F3	U	Unpacker encountered three 0xf3 words in a row	Silicon hybrid drops readout
Drop Readout (c2c2)	U	HDI readout length is 4 bytes long (0xc2c20000)	Silicon hybrid does not read out DOIM failure
Drop Readout (d0 ch)	U	Channel number 0xd0 = 208 is found in the data stream	OBDV is improperly clocked in by the FTM
Dropped Readout	U	EOR record found in the data stream before all chips have been unpacked	Silicon hybrid drops readout
Duplicate Chan (nn)	U	Duplicate channel number is found in the data stream (nearest neighbor unpacker)	Optical transmission error (*e.g.*, bit 0 is stuck)
Duplicate Channel	U	Duplicate channel number is found in the data stream ("classic" unpacker)	Optical transmission error (*e.g.*, bit 0 is stuck)
EOR Not Found	U	EOR record not found in the data stream	DAQ error
Event Count	S	SVXMon info message about the number of events with problems dumped to disk for a given run	End of a run
Event Sync Error	V	The "Event Synchronization Error" bit is set in the VRB error word of the SIXD/ISLD bank	VRB hardware error
Extra HDI	S	Some HDIs not included in the run (as marked in the hardware database) are found in the data stream	DAQ malfunction FIB id registers are not configured properly
ff or 7f Error	U	Channel byte differs from expected by more than one high bit	Optical transmission error Silicon chip hardware failure
Glink Frame Error	V	The "Glink Frame Error" bit is set in the VRB error word of the SIXD/ISLD bank	DAQ error
Glink Sync Error	V	The "Glink Synch Error" bit is set in the VRB error word of the SIXD/ISLD bank	DAQ error
Illegal Termination	U	HDI data stream termination character is neither 0xc1 nor 0xc2	Optical transmission error (bit 7 stuck high)

Message Type	Source	Meaning	Probable Causes
Internal Error	S	SVXMon detected an inconsistency between its data structures	Bug in SVXMon
Internal VRB Error	V	The "Internal VRB Error" bit is set in the VRB error word of the SIXD/ISLD bank	VRB hardware error
Invalid Cell Id	S	The cell id for a given chip is out of range (> 45)	Silicon chip hardware failure Optical transmission error
Invalid Chip Key	S	Invalid silicon chip identifier (SiChipKey) has been generated from chip position in the data stream	CDF software bug
Invalid EOR	U	Invalid word in the data stream in place where EOR record is expected	DAQ error
Invalid SRC Command	V	The "Invalid SRC Command" bit is set in the VRB error word of the SIXD/ISLD bank	DAQ error
Invalid Word	U	Generic bank unpacking failure	Optical transmission error DAQ error
Missing Chip ID	U	Chip id is not found in the data stream	Optical transmission error
Missing HDI	S	Some HDIs included in the run (as marked in the hardware database) are missing in the data stream	DAQ malfunction FIB id registers are not configured properly
No Data	S	No data unpacked for a given chip in several consecutive SVXMon events	The bank unpacker gives up because of unrecoverable errors upstream from the given chip The silicon configuration has changed, and SVXMon was not restarted The chip was intentionally "sparsified out" by setting high sparsification threshold in the initialization stream
No First Chip	U	First HDI data word is not a correct chip ID (and not wrong by just one bit)	Optical transmission error Silicon hybrid malfunction

Message Type	Source	Meaning	Probable Causes
No HDI Info	S	No HDI blocks inside VRB blocks of the SIXD/ISLD banks	DAQ failure
No Silicon Banks	S	No SIXD/ISLD banks in the event	Event builder failure
No VRB Info	S	No VRB blocks inside SIXD/ISLD banks	DAQ failure
No VRB Occupancies	S	No VRB occupancy data for this chip	DAQ or software failure
Out of Order Chip ID	U	Out of order chip id encountered in the data stream	Optical transmission error Incorrect chip id setting in the initialization stream
Pipeline Out of Sync	S	Chip pipeline cell id is different from the "most frequent" one	Synchronization loss on a chip due to noisy front end clock Readout chip hardware failure Optical transmission error
Run Config Invalid	S	Invalid chip key in the run conditions database	Database-related software bug
Strip Out of Order	U	Out of order channel number is found in the data stream	Optical transmission error
Stuck Bit	S	Stuck bit in the chip pipeline cell id	Optical transmission error
Stuck Cell Id	S	Chip pipeline cell id is stuck at a particular value	Readout chip hardware failure
Suspect Data	U	The unpacker made a wrong guess about channel numbering while trying to correct a bit error	Optical transmission error
TL1A Out of Sync	S	Time since Level 1 Accept is different for SIXD and ISLD banks	Reformatter error DAQ error
TS Out of Sync	S	Trigger supervisor counter is different for SIXD and ISLD banks	DAQ error
Unknown Channel	U	Fatal unpacker error at a channel number: not duplicate channel, not different from expected channel by one bit, and not all bits high	Readout chip hardware failure Optical transmission error

Message Type	Source	Meaning	Probable Causes
Unknown Chip Error	U	Fatal unpacker error at a chip ID: not the next chip ID, not different from expected by one bit, and not all bits high when a channel number is expected	Readout chip hardware failure Optical transmission error
VRBINFO_DEBUG_BX	S	An info message with a bunch crossing number for a given HDI	The "detailedVrbErrors" flag is turned on in the SVXMon configuration file
VRBs Out of Sync	S	Either the trigger supervisor counter or time since Level 1 accept is out of sync in several consecutive SVXMon events	See the description of "TL1A Out of Sync" and "TS Out of Sync" errors
Wrong Chip Count	U	The number of chips in the data stream is wrong for a given HDI	Optical transmission error
Wrong Data Size	S	Wrong data size for a given ladder in the VRB occupancy data	DAQ or software failure
Wrong HDI Count	S	The number of HDI blocks in the silicon banks has changed since the previous event	DAQ malfunction
Wrong VRB Count	S	The number of VRB blocks in the silicon banks has changed since the previous event	DAQ malfunction
*** Out of Limits	S	Some monitored quantity (occupancy, mean ADC value, RMS ADC value, *etc.*) is out of limits for a given chip. The type of the quantity is provided in place of ***. Please consult section 5 for further details.	High detector noise Optical transmission errors DAQ failure SVXMon is not configured to perform bad channel suppression

E Example Online Configuration File

```
#####################################################################
#                                                                   #
#       This is an example script to run SVXMon online.             #
#       It is a simplified version of the real thing.               #
#       It illustrates various settings which need to be            #
#       taken into account but skips complicated cut                #
#       definitions and histogram post-processing                   #
#       by SvxMonRunCompare.                                        #
#                                                                   #
#####################################################################

# Find an unused port for the error logger. This isn't a great
# solution because it may fail when a race condition exists
# between two processes, but it is better than doing nothing...
set unused_port 9101
while {[catch {socket -server {} $unused_port} channel]} {
    incr unused_port
    if {$unused_port > 65535} {
        puts stderr "ERROR in SVXMon configuration script:\
                out of TCP/IP ports"
        # AC++ redefines tcl built-in command "exit". Because of that
        # we have to use "exit" together with "return" in order to
        # terminate execution of this script.
        exit
        return
    }
}
close $channel

# Configure the set of AC++ modules used
path enable AllPath
foreach mod {
    TsAsciiInput
    PuffModule
    TestSiBanks
    MMFilter
    CT_Tracking
    SiClusteringModule
    SiPatternRecModule
} {
    module disable $mod
}

# Show currently used modules
```

```
module list
puts ""; flush stdout

# Get rid of the muon system. We don't run any muon reconstruction.
module talk GeometryManager
    DetectorMenu
        enableMuon set false
    exit
exit

# Point the calibration manager to the online database
module talk CalibrationManager
    add onotl_prd OTL cdf_reader/reader@cdfonprd
    DataDB      set onotl_prd
    Database    set onotl_prd
    ProcessName set L3_PHYSICS_CDF
exit

# Configure the Consumer Input module. Environmental variables
# CSLHOSTNAME and PARTITIONID will be set by the consumer framework
# scripts.
global env
module input ConsumerInput
mod talk ConsumerInput
   theCslHostname           set $env(CSLHOSTNAME)
   anyStreamBit             set true
   anyLevel3Bit             set true
   anyLevel2Bit             set true
   anyLevel1Bit             set true
   thePartitionId           set $env(PARTITIONID)
   isSilent                 set false
   debug                    set f
   veryVerbose              set f
   consumerType             set SVXMon
exit

# Output module configuration. This module is used to write
# events which illustrate silicon/DAQ problems. First, check
# if the environmental variable SVXMON_OUTFILE exists and
# use it as the output file name. If this variable is not set,
# make up a file name in the default location.
set default_output_dir /home/cdfdaq/SVXMon_events
set n_pass_max 0
if {[info exists env(SVXMON_OUTFILE)]} {
    if {$env(SVXMON_OUTFILE) != "0"} {
```

```
        set n_pass_max 2147483647
        set out_file_name $env(SVXMON_OUTFILE)
    }
}
if {$n_pass_max == 0} {
    if {[file isdirectory $default_output_dir]} {
        set n_pass_max 2147483647
        set out_file_name $default_output_dir/svxmon_bad_events.dat
    }
}
if {$n_pass_max > 0} {
    # Check that the file name starts with . or / -- this
    # is the DHOutput idiosyncrasy
    set firstchar [string index $out_file_name 0]
    if {$firstchar != "." && $firstchar != "/"} {
        set out_file_name "./$out_file_name"
    }
    # Check if the file already exists. Do not overwrite.
    if {[file exists $out_file_name]} {
        set version 1
        set newname $out_file_name.$version
        while {[file exists $newname]} {
            incr version
            set newname $out_file_name.$version
        }
        set out_file_name $newname
    }
    # Configure either DHOutput or FileOutput
    module output FileOutput
    module talk FileOutput
        dhCache set NONE
        output create badEvents $out_file_name
        output path badEvents AllPath
        output keepList badEvents \
            LRIH_StorableBank \
            SIXD_StorableBank \
            ISLD_StorableBank \
            SIXS_StorableBank \
            ISLS_StorableBank
        output list
    exit
}

# Set up some environment-related variables
if {![info exists env(CONSUMERBINDIR)]} {
```

```
    # The fall-back definition for the release directory
    # which contains the "Server" program
    set env(CONSUMERBINDIR) /data1/igv/consumer_b0/bin/$env(BFARCH)
}

# Where the histogram file will be written?
if {[info exists env(CONSUMERRESULTS)]} {
    set rootprefix "$env(CONSUMERRESULTS)/svxmon"
} elseif {[pwd] == "/data1/consumer/runarea/SVXMon"} {
    set rootprefix "/data1/consumer/results/SVXMon/svxmon"
} else {
    set rootprefix "svxmon"
}

# Which SVX Error Loger code we are going to run? Set up
# the environmental variable ERROR_LOGGER_DIR in order
# to use some version of the error logger which is
# different from the default.
set base_development "/cdf/code-Linux-2.0.34/cdfoffline/dist/releases/development"
set env(ERROR_LOGGER_DIR) "$base_development/SvxDaqMods/test/ErrorLogger"

# An option to disable the special SVXMon tcl prompt.
# May be useful to disable it for automatic SVXMon
# start-up/shut-down within the Consumer Framework
# system because this framework uses SIGINT to tell
# consumers when they should finish.
set disableTclPrompt 1

# Configure SVXMon
module talk SVXMonModule
  General
    # General program settings
    Debug_Print set false
    Start_Server set true
    CheckIfIntegrated set true
    if {$disableTclPrompt} {
        SmartPrompt set false
    }
    # Where to save histograms?
    RootFilePrefix set $rootprefix
    Save_ROOT_File set true
    # Options related to the error logger
    Start_Error_Logger set true
    Error_Logger_Exe set \
        "$base_development/SvxDaqMods/test/ErrorLogger/start_logger"
```

```
  Error_Logger_Port set $unused_port
  Error_Logger_Bufsize set 50
  # Send the Halt-Recover-Run requests?
  EnableHRR set true
  ErrorReceiverHost set "b0dap61.fnal.gov"
  ErrorReceiverPort set [expr 9040 + $env(PARTITIONID)]
  # How many events with problems to save
  MaxPassPerRun set 100
  MaxPassTotal set $n_pass_max
exit
SIXD
  Config_Script set {
    #
    # SIXD Monitor configuration. The commands below
    # will be evaluated in a separate Tcl interpreter.
    #
    # Get the list of bad strips from the database.
    # For simplicity, this example loads a fixed table.
    set loadBadStrips 1
    if {$loadBadStrips} {
        SIXDMon configure calibRunNumber 140287
        SIXDMon configure calibDataVersion 1
        SIXDMon configure calibStatus TEST
        puts -nonewline "Retrieving SISTRIPDH table\
             with run [SIXDMon cget calibRunNumber]\
             version [SIXDMon cget calibDataVersion]\
             status [SIXDMon cget calibStatus]... "
        flush stdout
        svx badstrips retrieve
        svx badstrips discard
        puts "Done"
        puts "[svx badstrips count] bad strips"
    }
    #
    # Parameters which configure the behavior
    # of Halt-Recover-Run requests
    SIXDMon configure chipsNeededForHRR 80
    SIXDMon configure hrrIntervalSec 300
    #
    # Error reporting mode. Set to 0 to report all errors normally.
    # Set to 1 to ignore some errors which happen simultaneously
    # with other errors (and caused by these other errors).
    SIXDMon configure reportingMode 1
    #
    # Histogram updating schedule
```

```
SIXDMon configure timeSoftUpdate 999999
SIXDMon configure timeHardUpdate 999999
SIXDMon configure neventsSoftUpdate 9999999
SIXDMon configure neventsHardUpdate 3
#
# Snapshot strategy for most plots. Should be set to 0, 1, or 2.
# 0 means no snapshots of the main strip dataset, 1 means one snapshot,
# and 2 means two snapshots.
SIXDMon configure snapshotStrategy 2
#
# Snapshot updating schedule. This schedule is used both for the
# main dataset of detector strips and for all 2D histograms. This
# means that a reasonable schedule should be specified even if
# the "snapshotStrategy" parameter is set to 0.
SIXDMon configure snapTimeSoftUpdate   9999999
SIXDMon configure snapTimeHardUpdate   9999999
SIXDMon configure snapEventsSoftUpdate 9999999
SIXDMon configure snapEventsHardUpdate 100
#
# Check VRB errors and pipeline cell ids
SIXDMon configure checkVrbErrors 1
SIXDMon configure detailedVrbErrors 0
#
# Number of events in each scan. Set to 0 if you
# don't need/want any scan plots.
SIXDMon configure scanPeriod 0
#
# SIXD Monitor will skip hits with pulse height below a certain
# threshold. Set this threshold low so that we always accumulate
# all hits.
SIXDMon configure pulseThreshold -300
#
# VRB occupancy monitoring. It is turned off here,
# so these settings are only an example.
SIXDMon configure monitorVrbOccupancy 0
SIXDMon configure vrbOccupancyHost b0dap30.fnal.gov
SIXDMon configure vrbOccupancyPort 9301
SIXDMon configure vrbOccupancySnapshots 1
#
# Print out the number of events processsed every 5 events.
# This code also updates the number of events in the error logger.
SIXDMon configure scriptPeriod 5
SIXDMon configure periodicScript {
    set numevt [svx nevents]
    errlog setsub periodicScript
```

```
    errlog ELincidental SVXMON_EVENT_COUNT $numevt
    puts "SVXMon processed $numevt events"
    flush stdout
}
#
# Creation of a slide show
svx folder "Slides" [list "* Chip Status Map"]
#
# Other HistoDisplayMain folders
svx folder "CO Folder (Silicon Raw Data)/Occupancy" \
        [list "Short-term chip occupancy, z-segment *"]
svx folder "CO Folder (Silicon Raw Data)/Mean Charge" \
        [list "Short-term chip mean charge, z-segment *"]
svx folder "CO Folder (Silicon Raw Data)/Charge RMS" \
        [list "Short-term chip rms charge, z-segment *"]
svx folder "CO Folder (Silicon Raw Data)/History" [list "History plot *"]
#
svx folder "SVXMon (expert)/Strip Plots 2D" [list "* vs strip # and *"]
#
# Cell id and strip plots. Need to make a lot of folders
# because of a bug in ROOT -- it can't display more than
# just a few hundred items in a folder
foreach {path pattern} {
    ISL/B5 {E [67] * -B *}
    ISL/B4 {E [67] * -A *}
    ISL/B3 {C [67] * -B *}
    ISL/B2 {C [67] * -A *}
    ISL/B1 {W [67] * -B *}
    ISL/B0 {W [67] * -A *}
    SVX/B5 {E [1-5] * -B *}
    SVX/B4 {E [1-5] * -A *}
    SVX/B3 {C [1-5] * -B *}
    SVX/B2 {C [1-5] * -A *}
    SVX/B1 {W [1-5] * -B *}
    SVX/B0 {W [1-5] * -A *}
    L00    {? 0 *}
} {
    svx folder "SVXMon (expert)/Strip Plots/$path" [list\
            "Number of readouts for $pattern strips"\
            "Strip occupancy for $pattern"\
            "Strip mean charge for $pattern"\
            "Strip rms charge for $pattern"\
            "Strip charge skewness for $pattern"\
            "Strip charge kurtosis for $pattern"\
            "Strip dnoise for $pattern"\
```

```
            "Bad strips for $pattern"\
            "Discarded strips for $pattern"\
            "New bad strips for $pattern"\
            "New good strips for $pattern"\
            "Strip VRB occupancy for $pattern"]
    svx folder "SVXMon (expert)/Cell Id Plots/$path" [list\
            "Cell id distribution for chip $pattern"]
}
#
# More folders
svx folder "SVXMon (expert)/Chip Plots" [list "Chip *"  "Median chip *" \
        "Short-term chip *"]
svx folder "SVXMon (expert)/Half-ladder Plots" [list\
        "Half-Ladder *"\
        "Short-term half-ladder *"]
svx folder "SVXMon (expert)/Layer Plots" [list "Layer *" "Short-term layer *"]
svx folder "SVXMon (expert)/Cell Id Plots" [list "*ell id distribution*"]
svx folder "SVXMon (expert)/SVT Plots" [list "SVT*"]
svx folder "SVXMon (expert)/Tracks" [list "Average_Residuals*" \
        "Tracks_in_Half-Ladder*"]
svx folder "SVXMon (expert)/Other Plots" [list "*"]
#
# Actual creation of monitoring plots
#
# Silicon tracks
set track_handle [histo tracks SVT]
foreach {parameter value} {
    nPhiHitMinG 5
    nPhiHitMinB 3
    nZHitMinG 0
    nZHitMinB 0
    ptMinG 1.0
    ptMinB 0.5
    ptMaxG 100.0
    ptMaxB 100.0
} {
    $track_handle configure $parameter $value
}
#
# Although somewhat redundant, the ladder histograms are useful
# because one can quickly look up which ladders are integrated
histo ladders {occupancy mean stdev} 0
#
histo chips {occupancy mean stdev} 1
histo layers phi {occupancy mean stdev} 1
```

```
histo layers z {occupancy mean stdev} 1
histo occupancy 100 log linear 1
#
# Detectors included in the silicon system for this run
set detector_list [svx numerology [svx getrun]]
#
# Check that all ladder names are correct. This is
# needed because of some problems with SOC which is
# not supported by the standard silicon numerology classes.
set detector_list_filtered {}
foreach detector $detector_list {
    if {[llength $detector] == 5} {
        lappend detector_list_filtered $detector
    } else {
        puts stderr "WARNING: bad ladder specifier \{$detector\}"
    }
}
set detector_list $detector_list_filtered
#
foreach detector $detector_list {
    histo strips $detector {occup mean stdev} 1
}
#
set special_detector_group [lrange $detector_list 0 2]
#
foreach detector $special_detector_group {
    histo daqparam bunchx $detector {nhits mean stdev} 0
    histo strips2d cellid $detector {nevents mean stdev} 0
    histo strips2d adc $detector {nevents} 0
}
#
# Cell id histograms
foreach detector $detector_list {
    set nchips [svx numchips $detector]
    for {set chip 0} {$chip < $nchips} {incr chip} {
        histo cellid $detector $chip
    }
}
#
# Pipeline cell id checker and status map
histo cellid global
set cellid_handle [histo cellid map]
$cellid_handle configure hysteresis 5
$cellid_handle configure needEvents 5
#
```

```
# Data quality checker
set check_handle [histo check {occup mean stdev} 1]
#
# First, disable checks for all detectors
foreach detector [svx numerology all] {
    set nchips [svx numchips $detector]
    for {set chip 0} {$chip < $nchips} {incr chip} {
        $cellid_handle configure skipChip($detector,$chip) 1
        $check_handle configure skipChip($detector,$chip) 1
    }
}
#
# Now, enable checks and setup cuts
# for the sensors included in the system
foreach detector $detector_list {
    set nchips [svx numchips $detector]
    for {set chip 0} {$chip < $nchips} {incr chip} {
        $cellid_handle configure skipChip($detector,$chip) 0
        $check_handle configure skipChip($detector,$chip) 0
        # The following cuts are rather simplistic. The real
        # SVXMon configuration file would use more complicated,
        # layer and sensor side dependent cuts.
        foreach {cut_name run_value snapshot_value} {
            occup_nsigma_high  10.0     10.0
            occup_nsigma_low   10.0     10.0
            occup_limit_high   101.0    101.0
            occup_limit_low    0.5      0.5
            mean_nsigma_high   50000.0  50000.0
            mean_nsigma_low    50000.0  50000.0
            mean_limit_high    300.0    300.0
            mean_limit_low     -100.0   -100.0
            stdev_nsigma_high  50000.0  50000.0
            stdev_nsigma_low   50000.0  50000.0
            stdev_limit_high   300.0    300.0
            stdev_limit_low    -2.0     -2.0
        } {
            $check_handle configure \
                    chipcut($cut_name,0,$detector,$chip) $run_value
            $check_handle configure \
                    chipcut($cut_name,1,$detector,$chip) $snapshot_value
        }
    }
}
#
# Chip groups. Combine chips which have the same
```

```
# layer, side, DPS setting, and readout mode
# (read all, nearest neighbor, etc.) into groups.
# Then we can compare chips within each group
# to each other.
set run [svx getrun]
foreach detector $detector_list {
    set nchips [svx numchips $detector]
    foreach {barrel ladder_seg phi_wedge layer side} $detector {}
    for {set chip 0} {$chip < $nchips} {incr chip} {
        set dpsmode [svx dpsmode $detector $chip $run]
        set readmode [svx readmode $detector $chip $run]
        lappend chipgroups($layer,$side,$dpsmode,$readmode) \
                [list $detector $chip]
    }
}
set groupname_list {}
foreach l_s [lsort [array names chipgroups]] {
    foreach {layer side dpsmode readmode} [split $l_s ,] {}
    # The group name must not have a comma inside because
    # of how the group cuts are parsed.
    if {$readmode == 0} {
        if {$dpsmode} {
            set groupname "L$layer $side side readall"
        } else {
            set groupname "L$layer $side side nodps readall"
        }
    } else {
        if {$dpsmode} {
            set groupname "L$layer $side side"
        } else {
            set groupname "L$layer $side side nodps"
        }
    }
    lappend groupname_list $groupname
    $check_handle chipgroup set $groupname \
            $chipgroups($layer,$side,$dpsmode,$readmode)
    #
    # Cuts on medians for the groups. In this simple
    # example we allow everything to pass, so that
    # group-level error messages are never generated.
    foreach {cut_name run_value snapshot_value} {
        occup_high  101.0    101.0
        occup_low   -1.0     -1.0
        mean_high   256.0    256.0
        mean_low    -30.0    -30.0
```

```
        stdev_high  100.0    100.0
        stdev_low   -1.0     -1.0
    } {
        $check_handle configure \
                groupcut($cut_name,0,$groupname) $run_value
        $check_handle configure \
                groupcut($cut_name,1,$groupname) $snapshot_value
    }
}
#
# Build the chip group canvases after all groups have been configured
$check_handle configure labelSize 0.04
$check_handle configure padBottomMargin 0.33
$check_handle plotgroups
#
# Lower limits on various quantities. Useful to prevent
# chips with small problems from failing. For example,
# in read-all mode the most common occupancy is 100%
# and the range is 0%. To allow chips with 99.9999%
# occupancy to pass, we limit the range from below.
foreach {name value} {occup 1.0 mean 1.0 stdev 0.2} {
    $check_handle configure minimalRange($name) $value
    set inst "inst $name"
    $check_handle configure minimalRange($inst) $value
}
#
# The number of events needed to produce the "No Data" errors
$check_handle configure maxEventsNoLadder 50
$check_handle configure maxCallsNoLadderVRB 50
#
# The number of events needed to produce various "Out of Limits"
# errors for the chips. This number should normally be smaller
# than the number of events in a snapshot in case snapshots
# are requested in the "histo check" command.
$check_handle configure needRunEventsChips 50
$check_handle configure needSnapshotEventsChips 50
#
# Same for group quantities
$check_handle configure needRunEventsMedians 50
$check_handle configure needSnapshotEventsMedians 50
#
# Lower the error severity levels if bad strips are not loaded
if {!$loadBadStrips} {
    foreach {errtype level} {
        "Occup Out of Limits" "ELincidental"
```

```
        "Mean Out of Limits"  "ELincidental"
        "Stdev Out of Limits" "ELincidental"
        "Inst occup Out of Limits" "ELincidental"
        "Inst mean Out of Limits"  "ELincidental"
        "Inst stdev Out of Limits" "ELincidental"
    } {
        $check_handle configure errorSeverity($errtype) $level
        $check_handle configure chipSeverity($errtype) $level
    }
}
#
# Make history plots for group quantities. Up to
# 14 groups can be displayed on one history plot.
# We use 8 to balance the number of graphs on each canvas.
set groups_per_plot 8
set n_plots [expr [llength $groupname_list] / $groups_per_plot]
if {[expr [llength $groupname_list] % $groups_per_plot] > 0} {
    incr n_plots
}
foreach {plottype title} {
    occup "Median chip occupancy"
    mean  "Median chip readout charge (ADC Counts)"
    stdev "Median chip RMS charge (ADC Counts)"
} {
    set root_histo_name "chip group histo $plottype"
    set groupnumber 0
    for {set plotnumber 0} {$plotnumber < $n_plots} {incr plotnumber} {
        flush stdout
        set subgroup [lrange $groupname_list \
                [expr {$plotnumber * $groups_per_plot}] \
                [expr {($plotnumber + 1) * $groups_per_plot - 1}]]
        set handle_list {}
        foreach groupname $subgroup {
            lappend handle_list [svx watch $root_histo_name \
                    bin $groupname $groupnumber]
            incr groupnumber
        }
        histo history $title $handle_list
    }
}
#
# Get a timestamp to measure the wall-clock code speed
set begin_time [clock seconds]
#
# Tcl script executed at the end of job
```

```
      SIXDMon configure endJobScript {
          # Update the number of events for the error logger
          errlog setsub endJobScript
          errlog ELincidental SVXMON_EVENT_COUNT [svx nevents]
          #
          # A real online SVXMon configuration file would run
          # SvxMonRunCompare at this point
          #
          set interval [expr [clock seconds] - $begin_time]
          set hrs [expr $interval / 3600]
          set min [expr ($interval - $hrs * 3600) / 60]
          set sec [expr $interval - $hrs * 3600 - $min * 60]
          puts "SVXMon running time $hrs hrs $min min $sec sec"
          flush stdout
      }
      # Do not clean up all root objects in the destructor.
      # The complete cleanup is very slow because root's
      # "garbage collection" mechanism is extremely inefficient.
      SIXDMon configure destructAll 0
      #
      # Limits on the number of dumped events
      SIXDMon dumplimit "VRBs Out of Sync" "Silicon/DAQ" 5
      SIXDMon dumplimit "Bunch X Out of Sync" "Silicon/DAQ" 5
      foreach {errorType limit} {
          "Missing Chip ID"      5
          "Strip Out of Order"   5
          "Out of Order Chip ID" 5
          "Invalid EOR"          5
          "EOR Not Found"        5
          "Data Found After EOR" 5
          "Bad Chip ID"          5
          "Wrong Chip Count"     5
          "Pipeline Out of Sync" 2
          "Dropped Readout"      5
          "Invalid Word"         5
          "Chip Side Error"      5
          "Duplicate Channel"    5
      } {
          SIXDMon dumplimit $errorType $limit
      }
    }
  exit
exit

# Start running
```

```
ev begin

# The following cycle is a workaround for the buggy
# interrupted system call handling by the CSL library.
# The cycle will be terminated when the global variable
# svxmon_read_data is set to 0. This happens if the user
# types "stop" at the SVXMon tcl prompt.
if {!$disableTclPrompt} {
    global svxmon_read_data
    while {$svxmon_read_data} {
        ev continue
    }
}
exit
```

F Example SvxMonRunCompare Parameter File

```
// This is a lookup table for comparison parameters
// used by SvxMonRunCompare.
// Format for parameters:
//      quantity parameter_name value
// Format for mailing list:
// mail u1@fnal.gov u2@fnal.gov ...
// Format for turning on or off the pop up alarm:
// pop_alarm t     // pop up alarm is on
// pop_alarm f     // pop up alarm is off
// (words separated by spaces only; double slashes "//" are commented)
//
// Author: Henri Bachacou, bachacou@fnal.gov (2002/04/02)

// Parameters for ``chip with failure'' check:
mean nsigma_high            7.
mean nsigma_low            -7.
mean limit_high           300.
mean limit_low              0.
// Parameters for ``unbiased chip'' check:
mean bias_signif_high     300.
mean bias_signif_low       -4.

// Same for other variables...
stdev nsigma_high           7.
stdev nsigma_low           -7.
stdev limit_high          300.
stdev limit_low             0.
stdev bias_signif_high    300.
```

```
stdev bias_signif_low      -4.
//
stdevVar nsigma_high        7.
stdevVar nsigma_low        -7.
stdevVar limit_high       300.
stdevVar limit_low          0.
stdevVar bias_signif_high 300.
stdevVar bias_signif_low   -4.
//
occup nsigma_high           7.
occup nsigma_low           -7.
occup limit_high          101.
occup limit_low             0.
occup bias_signif_high      6.
occup bias_signif_low    -300.

// Email list for alarm report:
mail bachacou@fnal.gov

// Turn on the popup window alarm:
pop_alarm t
```

References

[1] Hans Wenzel, Kaori Maeshima, "Online monitoring for Run II", CDF/DOC/ONLINE/PUBLIC/4835.

[2] http://www-cdfonline.fnal.gov/consumer/home/consumer_home.html

[3] Brent Welch, "Practical Programming in Tcl and Tk", 3rd Edition, Prentice Hall, 2000.

[4] http://cdfcodebrowser.fnal.gov/CdfCode/source/ErrorLogger/doc/html/0ErrorLogger.html

[5] http://sourceforge.net/projects/blt/

[6] A really small package for accessing UNIX system services "signal" and "waitpid" from tcl. Available from http://www-cdf.lbl.gov/~igv/tcl_extensions/signal_mod1.4.tar.gz.

[7] http://nkek15.fnal.gov/cgi-bin/newproblemdb/DBmain.pl

[8] Many SVXMon commands take an argument (usually called *detector* in this note) which specifies one side of a silicon half-ladder. This argument is a five-element tcl list {*barrel ladderSegment phiWedge layer side*}. Allowed values for *barrel* and *ladderSegment* are "west", "east", and "center". Allowed values for *side* are "z" and "phi". *layer* and *phiWedge* are integers. *layer* should be set to 0 for L00, to 1 for SVX layer 0, and so on, up to 7 for the outer ISL layer.

[9] We use the term *handle command* or simply *handle* to describe tcl commands which provide an interface to functions of a particular object rather than a whole class of objects. A typical usage of an SVXMon handle is "`$handle configure parameterName`" or "`$handle cget parameterName`" where "parameterName" is a name of some parameter-like object member.

[10] E. Sexton-Kennedy and M. Shapiro, "Creating Talk_to Menus for AC++", CDF/DOC/COMP_UPG/PUBLIC/4521.

[11] I. Volobouev, "Speaking Better Tcl to Your AC++ Modules", CDF/PUB/COMP_UPG/PUBLIC/5731.

[12] http://www-cdfonline.fnal.gov/cdfdb/servlet/RunTypes

[13] M. Bishai *et al.*, "An SVX3D Chip User's Companion", CDF/DOC/TRACKING/GROUP/5062.

[14] W. Badgett *et al.*, "Description of the CDF Data Structure for Run II", CDF/DOC/CDF/PUBLIC/4152.

[15] J. Conway *et al.*, "Numbering and Naming Convention for SVX II and ISL", CDF/DOC/TRACKING/PUB/4103.

[16] http://www-cdf.lbl.gov/~bachacou/form.html

[17] http://tcl.activestate.com/man/tcl8.0/TclCmd/string.htm#M10

[18] http://fcdfhome.fnal.gov/usr/bachacou/SVXMon/SVXMon_CO_manual/svxmon_CO_instructions.html