A 1Ghz digital Oscilloscope in (mostly) Tcl

Ronald Fox
National Superconducting Cyclotron
Laboratory, Michigan State University
640 S Shaw Lane
East Lansing, MI 48824-1321
CAEN Technologies

1140 Bay Street Suite 2C
Staten Island, NY 10305
ron@caentech.com

ABSTRACT

The transition of nuclear and particle physics data acquisition systems to high speed wave form digitization technology allows a great deal of simplification in electronics setups. There are trade-offs however. So-called 'digital data acquisition systems' require higher data transmission bandwidths and can be challenging to set up for the uninitiated.

This paper will address a software product that is being developed to address the first stages of setup. In this work the large and unfamiliar parameter space of the waveform digitizer is hidden behind a cognitive model that is familiar to most experimenters, that of a digital oscilloscope.

1. INTRODUCTION

Traditional nuclear and particle physics data acquisition systems made use of 'feature digitizers'. These devices perform an analog analysis of a waveform and provide a single value to the host/readout software. For example a Wilkinson ADC captures the height of the waveform peak and provides a number that is proportional to that height.

Traditional nuclear and particle data acquisition systems often require a significant investment in analog electronics to condition the signals presented to the digitizer. For example to use a TDC (Time to Digital Converter), an external discriminator is required to convert both the inputs and some reference signal to digital signals.

In recent times, however, improvements in the performance of high speed flash ADCs (FADCs) and high capacity field programmable gate arrays (FPGAs) have ignited a transition from feature digitizers to the use of waveform digitizers coupled with FPGAs in which are embedded digital signal processing algorithms. Indeed there are detector systems such as arrays of segmented germanium detectors which cannot be analyzed fully in any other way.

In experimental Nuclear Physics data acquisitions systems based around these FADC/FPGA combinations are called "Digital Data Acquisition Systems" (DDAS).

These advances do not come without a cost, however. During at least some part of the system setup, and during system diagnostics, it is common to need to transmit raw waveform data from the front end electronics to the online diagnostics software. This requirement drastically increases the bandwidth requirements of these systems since typical digitization speeds may exceed several hundred MHz.

Furthermore, to productively take data from digital data from a DDAS a large number of parameters must be optimized for each channel. These parameters range from those that allow the raw signal to be properly digitized to parameters that affect the digital signal processing performed by the FPGA firmware associated with each ADC channel. This can be challenging and frustrating for users new to the DDAS paradigm.

This paper will describe a project that provides researchers with a simplified familiar cognitive model that makes one set of these parameters more approachable. The scope of this project is limited to helping researchers set up the digitizer to properly accept waveforms from their detectors.

It turns out that early stages of the setup of a digitizer channels are analogous the process of

setting up an oscilloscope. Therefore we provide users with digital oscilloscope software that operates by setting appropriate digitizer parameters.

This software is largely written in Tcl/Tk. The paper will discuss:

- The cognitive model being provided to the user
- The wrapping of the C API with a Tcl command ensemble, and how digitizer model differences are handled.
- The considerations behind the choice of a graphics package and its implications.
- The application's functionality and user interface.
- Issues uncovered and solutions arrived at in packaging the software in Starpack format.
- Future work.

2. COGNITIVE MODEL

In past efforts by CAEN to provide support for visualizing waveforms, presented the digitizer parameters to he user without interpretation. These parameters have names like "post-trigger", "trigger level", channel mask, and sample window. New users were often confused about the meaning of parameters and how each parameter impacted data acquisition. As such users often had trouble getting meaningful waveforms to appear in the waveform display.

Acquiring and displaying waveforms is, however something that most users have done in the past. Almost all users were knew how to set up and use a digital oscilloscope to display the signals that they then were going to digitize with their ADC.

We therefore decided to try to model the software after the controls in a digital oscilloscope. The detailed digitizer parameters and their internal names could then be hidden from the user behind more familiar controls such as trigger threshold, channel select, trigger delay.

We were also tempted to add a setup wizard, however wizards can be frustrating to users who already know how to use these devices. What we did instead was provide a guide at the bottom of the oscilloscope user interface led inexperienced users through the process of setting up the digitizer while allowing experts to bypass steps in the process, or the entire process. At each step, a subset of the 'scope' controls are enabled, while in the 'expert' mode all controls are enabled.

3. THE API ITS WRAPPER AND MORE

CAEN provides a layered set of libraries[1] that hides most of the model-to-model differences from the programmer. This allows digitizers to be programmed at a basic level without caring which digitizer the application is using.

The first step of producing the scope application was to produce a Tcl wrapping of the digitizer libraries. The libraries include data types that are not suitable to a swig wrapping, so this wrapping was done by hand. The wrapper is built on top of both the digitizer library and the Tcl++ library that was described elsewhere [2].

The wrapper library appears to the Tcl programmer as a command ensemble in the **::CAEN::** name space with the base command **digitizer**. Sub commands each wrap a specific digitizer library function. The parameters of each function have, where necessary been made Tcl friendly, for example enumerated types are converted back and forth between string representations (e.g. a value like CAEN_DGTZ_XX724_FAMILY_CODE which is a value in the digitizer family enumerator is mapped in Tcl to the string "xx724".

3.1 Module capabilities

While the CAEN digitizer libarary does a good job of insulating the programmer from most model specific issues, some model-to-model differences are not, however, transparent to the programmer. These include but are not limited to:

- Connection between the host and the digitizer.
- Sampling frequency of the digitizer
- Buffer organization of the digitizer.
- Number of channels.
- Input voltage ranges

The API does provide a mechanism to query the digitizer model family and model version (a model version is a particular digitizer model within a family). With the exception of the input voltage range, the model family and model version are sufficient to select the properties of a module.

3.2 Representing module capabilities

Module capabilities are stored in an SQLite database. A package in the software can convert database information for a module to a dict. Child records of a module's master record are represented within the dict as lists of values or sub-dicts depending on whether or not the sub-record has multiple fields that need to be represented.

A small Tk application has also been written to maintain the module database. This application allows the database to be maintained by device designers.

3.3 So you want to take data too?

The digitizer modules have multi-event memories. While some connection types (e.g. CONET and CONET connected VME bus interfaces) support interrupts, others (USB Connected VME bus interfaces and desktop digitizers with USB interfaces) do not and must be polled.

Furthermore the API presents interrupts to the user as events which must be polled. Thus the intent is for the API to be polled by the application for events which then are read out of the digitizers.

This model provides three implementation alternatives within a Tcl/Tk application driven by an event loop:

1. Drive the poll from a self-resetting **after** timer, or the libTcl API equivalent (Tcl_CreateTimerHandler).
2. Create a new event source for the Tcl/Tk event loop.
3. Run the polling in a thread and post and event to the main application thread when data are ready. This can be done either at the C level or via the tcl thread package.

The alternative chosen was to use the Tcl thread package to implement option 3. Since the CAEN digitizer library is thread safe this option does not pose too much difficulty, except for the edge case of application exit with the trigger loop active.

The trigger thread is therefore structured as a one shot. When the application is able to respond to a trigger, It tells the trigger thread to begin polling. The polling thread then polls for data available and posts and event back to the main thread when data is available. The main thread processes the event and then signals the poll thread to resume trigger polling.

When the process exit is requested, if data taking is enabled, a software trigger is generated to ensure the polling thread sees data. Once that trigger is posted back to the main thread, the handles open on the digitizer can be closed and the exit operation completed.

4. REAL-TIME GRAPHICS.

The scope program is a graphically intensive program. For each trigger, the program must draw as many as 8 traces of typically several thousand points each. At high rates, it is not necessary that the software capture every trace, however the updates appear smooth and the user interface must be responsive.

The following packages were considered for trace plotting:

1. BLT

2. Refactored BLT Components (RBC).
3. Plotchart
4. Roll-your-own canvas graphics.
5. The remainder of this section will briefly describe each of these packages, the pros and cons of using them within this application. A concluding sub-section will describe the choice and what was needed to get the choice to work.

4.1 BLT

BLT is a Tcl/Tk extension written and maintained by George Howlett [3]. It is a complied extension that features a wide variety of sophisticated graphical components.

Pros:

1. As a compiled extension its performance is very good.
2. Its design lends itself easily to time varying graphics. Each trace could be represented as a vector and updating the plot would simply be a matter of updating the points in the vector.
3. Support for expansion, coordinate transformation and feature picking is very good.
4. The author has used BLT in other work and was familiar with it.

Cons:

1. BLT is not stubs enabled and the intent from the beginning was to embed this application in a starpack. Note however that TclKits that contain BLT do exist.
2. BLT has not kept up with Tcl/Tk. While patches exist to make it work on Tcl/Tk 8.5, and while the repository shows that work is being done on the project (last update at the time this paper was going through first draft was April 30, 2013), it is not clear that the software author is able to commit the resources needed to continue to evolve the software.
3. BLT has a few dirty hooks into Tcl/Tk which would make 'self-maintenance' of the software difficult in case the project were abandoned.
4. As a compiled extension (I know I called this an advantage earlier), embedding the software in a Starkit/Starpack results in some challenges (see section 6).

4.2 Refactored BLT Components

The Re-factored BLT Components project (RBC) [4] is a project that is run by Bob Techentin. It is a fork of the BLT project. The intent of the project was to remedy the short-comings of BLT by producing a stubs compatible BLT that was decoupled from the internals dependencies that the original BLT suffers from.

Pros:

1. RBC does work with Tcl 8.5 (this author has no experience attempting to use RBC with 8.6 releases although the project pages says that it has been used with 8.6 beta releases).
2. The author has used parts of RBC with a project that used BLT and needed to migrate to Tcl 8.5 with success.
3. RBC is highly compatible with BLT so the author's knowledge of BLT could be used directly with RBC.

Cons

1. Bob classifies the software as "ready for an alpha release" which indicates there may be some pitfalls if used in a demanding environment.
2. RBC still has linkages to Tcl/Tk internals.
3. RBC has not yet been tested with Starkit/Starpacks.
4. The most recent set of releases dates back to late 2009 and the most recent set of commits were documentation commits in 2011. Unfortunately this project seems likely to be dead.

4.3 Plotchart

Plotchart [5] is a pure Tcl scientific plotting package written by Arjen Markus. Plotchart is distributed as part of the Tklib.

Pros

1. Arjen is very responsive to questions about Plotchart and very willing to make changes and bug-fixes on request.
2. Since Plotchart is pure Tcl/Tk, self-maintenance of the package in the event Arjen is not able to continue his work on it is easier.
3. Since Plotchart is a pure Tcl/Tk package, including it in a Startkit/Starpack is trivial.
4. Plotchart provides primitives for coordinate transforms that, together with event handling can add the ability to interact with the plot.

Cons

1. Plotchart's pure Tcl nature (I know I said this was an advantage too) leads to concerns about performance.
2. Plotchart's focus is on static graphics. As such it is not clear it can be easily adapted to Scope's need for dynamically changing graphics (See section 4.5 below).
3. This author was not familiar with Plotchart and therefore there would be a learning curve to using it effectively.

4.4 Roll Your Own Canvas Graphics

This option means creating my own graphics package designed for and adapted to the needs of the scope program.

Pros:

1. The resulting package will be well suited to my needs.
2. The resulting package will be something I can easily understand and be maintainable by me.

Cons

1. Time spent creating and debugging this package would detract from time that could be spent on the main application.

4.5 Decision and Consequences

None of these choices is ideal. The most attractive of all of them, however was Plotchart. Some performance testing was done by plotting a randomly phase shifted sine wave and a randomly phase shifted saw-tooth to see how quickly Plotchart could update the plot.

Plotchart's performance was deemed acceptable, although a couple of emails back and forth with Arjen were needed to fix some issues with the package that popped up as a result of this test.

As stated in section 4.3, Plotchart is not really designed for the dynamic graphics that would be produced by the scope application. For example, Scale changes that, in BLT are performed by simply changing options in the plot widget, require a complete regeneration of the plot.

As development progressed, it was useful to encapsulate the plots used by Scope in a SNIT [6] container that would transparently recreate the plot as needed, as well as provide other services required by the scope application such as markers and a key of markers. At the end of the day, however I think Plotchart was a good choice.

5. Functionality and User Interface

This section will describe the user interface of the scope program in some detail.

When starting the program, the first thing that must be provided is information about the connection to the digitizer. This is currently done via a dialog shown in Figure 1 below. The drop down menu on the left allows the user to select between the communications interfaces available and provide parameters that allow the specific digitizer module to be selected from the set attached to that interface.

Once connection has been established with a digitizer, the software reads its model family and model instance and looks up its characteristics in an SQLite database. The database is part of the Starpack's VFS, however it is copied into a temp directory based on where the operating system likes to put such things. See future work however.

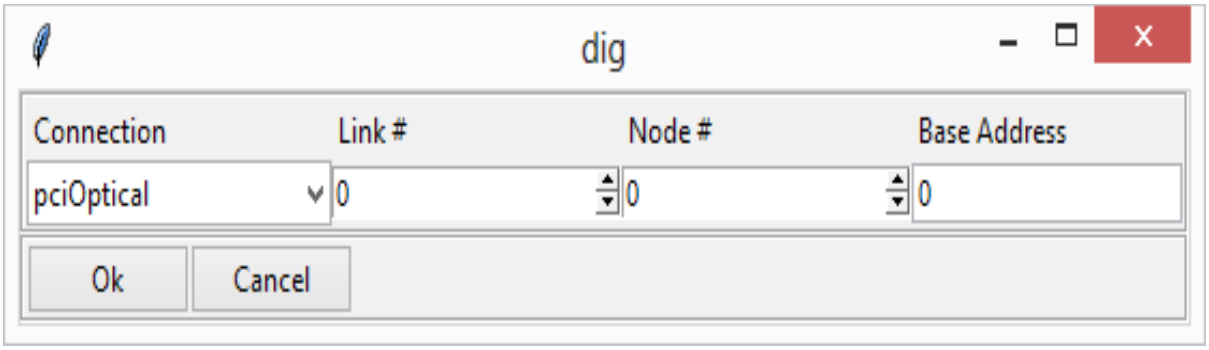

Figure 1: Connection parameter prompt

Unfortunately, the input voltage range of the digitizer is an option that cannot be determined. For each model family and model instance, there are a set of possible input ranges the user specifies when ordering the module.

The model family, model instance and serial number uniquely identify a module. The serial number is also read from the device. These are used to attempt to locate the module in another SQLite data base that is stored in the user's home directory tree (again the exact location of this database depends on the operating system). If the module has been used before, its database record provides a voltage range as a low and high limit pair.

If the module has never been seen before, the set of possible voltage ranges are presented and the user must select the correct voltage range for the module. This information is stored in the user's database and the same user will never again be prompted for this information.

Figure 2 below shows the full user interface. In this section I will describe in turn:

- The channel and trigger selection panes,
- The Scope controls pane
- The Guide pane
- The ways in which the user can interact with the plot itself.
- The menu commands.

5.1 Channel and trigger selection

The number of channels is one of the properties of the module family/module instance. The database lookup of this information is used to generate the grid of channel selection and trigger selection checkboxes and trigger selection. While the digitizer is continuously sampling, an external trigger and channel triggers are used to determine when to store a trace for readout by the host computer. The external trigger is exactly that, a logic input whose falling edge is used to indicate a waveform should be stored. The channel triggers are driven by the FPGA firmware associated with each channel. These provide a leading edge discriminator which can be set to fire when the rising or falling edge of a channel crosses a threshold value. The scope controls section (5.2) will provide more information about how the trigger parameters are set.

Triggers can only come from checked channels in the trigger selection pane. Note that this is a hardware/firmware trigger enable, not a software filter from amongst a more inclusive set of triggers. The module trigger is the logical or of all contributions to the trigger.

When the scope is triggered, only the checked channels are plotted. Where possible, the module is programmed to disable the transfer of unchecked channels to optimize data transfer performance.

In addition to the external and channel hardware triggers, the software can force a trigger at any time. While this feature seems useless, its purpose and usefulness will become clear in the subsequent sections.

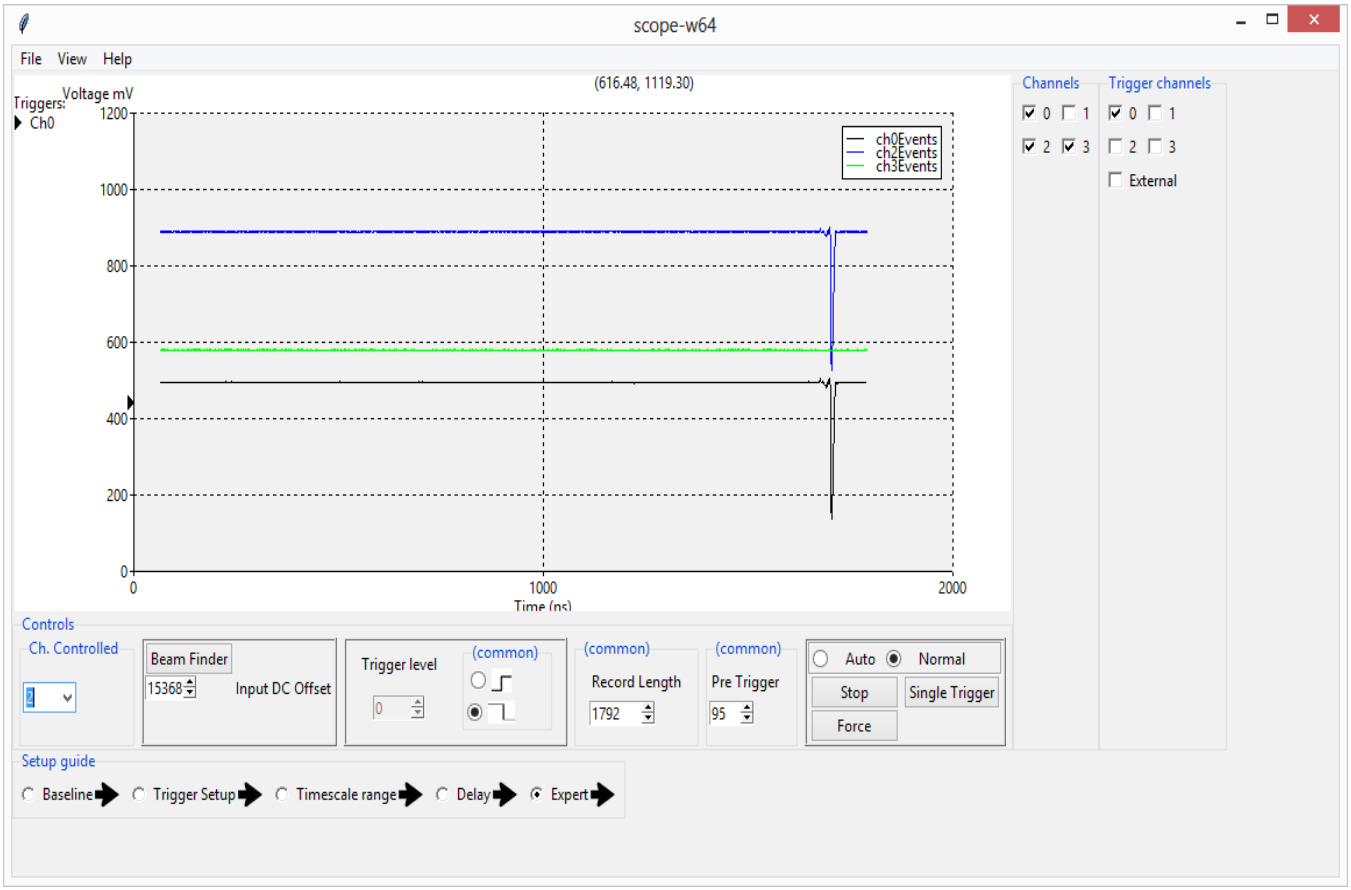

Figure 2

5.2 Scope controls

This section looks at the scope controls. For the most part, scope controls are laid out from left to right in the order in which they will be used. The procedure guide pane (see section 5.3) can further disable the set of controls that are not relevant to the operation being performed.

The left most control is a channel selection control. Most controls operate on a per-channel basis. Those that do not are labeled as common on the user interface. The selected channel determines which channel is selected for control.

5.2.1 DC Offset control

The left most box of controls is used to inject a DC offset on the signal. This is analogous to the vertical position knob on an analog/digital scope. By adjusting this value the signal can be raised or lowered within the voltage acceptance region of the digitizer. Note that each channel has an independent DC offset adjust.

Often when first plugging a signal into the scope, the user does not have a good idea where the baseline is in the window. This is due to DC offsets on the signal itself. The **Beam Finder** button attempts to automatically set the channel DC offset so that the signal baseline is located in the center of the acceptance window.

The beam finder operates by performing several software triggers. The idea is that these are asynchronous with the actual signal and are therefore likely to result in one or more traces that are largely baseline. The average value of several representative points is computed and, if there are significant outliers from this average they are discarded and the average repeated with the reduced data set. This average is again averaged over all software triggers and the DC offset required to center a trace point with that value is computed and programmed.

In practice this simplistic beam finder algorithm works quite well.

5.2.2 Trigger edge and level

To the right of the DC offset controls are the trigger controls. These control the direction and threshold for the leading edge discriminator.

Each channel has a leading edge discriminator. This discriminator is what determines when a channel triggers the digitizer to store a trace. The edge direction (rising or falling) determines whether the trigger occurs when the waveform goes below (falling edge) the trigger value or above (rising edge). The edge direction, as shown is common for all channels. The external trigger is a logic input and therefore has no threshold associated with it.

In keeping with the guide philosophy, the trigger controls are only enabled if the selected channel is enabled to contribute to the trigger.

Markers to the left of the Y axis display the trigger threshold for each channel that contributes to the trigger. A key at the top left links the color of each marker to the channel it controls. The marker color also matches the color of the corresponding channel's trace if that channel is enabled for display (it is legal to trigger on a channel without looking at it).

5.2.3 Record Length Controls

To right of the trigger level and edge adjustment is the record length control. This control is common across all channels of the digitizer.

Each digitizer module has a large memory dedicated to on-board event storage. The buffer organization of that memory is programmable. One can trade off the number of samples acquired per trace with the number of events the module can buffer before it is not able to respond to triggers.

On an oscilloscope adjusting the per event buffer size (record length) corresponds to adjusting the full scale time of the scope display.

As with a digital scope, since increasing the record size does not change the sampling frequency one does not change the signal resolution when adjusting this parameter.

5.2.4 Pretrigger

The digitizer modules are continuously digitizing into a ring buffer. A trigger simply tells the digitizer firmware to continue digitizing in a different ring buffer until the next trigger occurs.

The actual 'closing' of an event's buffer can occur at a programmable time after the trigger. This is useful because normally the trigger indicates the beginning of an event of interest rather than the end of an event of interest. The pretrigger adjust (common to all channels), corresponds to a delay after trigger adjustment on a digital or analog scope.

Adjusting this can position the part of the trace of interest anywhere in time within the time window captured by the trace. The ability to look back in time also provides the ability to see the signal prior to the actual trigger condition. This can be useful for signals with a slow rise-time that contain useful information on the leading edge of the signal.

One common use of this in nuclear physics is the use of fast/slow scintillator pairs to do particle identification. In that case one has a single photomultiplier on the back of a pair of detector crystals that react to ionizing radiation by emitting light. One crystal's properties may create a prompt signal while the other may create a much slower signal. Normally the slower signal provides something like the total energy while the prompt signal provides good timing (for the end of a time of flight leg for example). One can then use the discriminator to provide a low energy cut off but nonetheless look back to capture the timing of the prompt signal from the faster part of the detector.

5.2.5 Data Acquisition Control

The final control (rightmost) controls data acquisition. This section of the controls panel attempts to address several competing needs:

1. When first setting up it is normal to not really know what the trigger level should be or where the signal baseline will actually wind up (especially if you don't do a beam find.
2. Sometimes, when the trigger fails its good to see something even if it's not a meaningful signal.
3. When the trigger is set up properly it should be the only reason traces are saved by the module and read from the module.
4. There are times when you want to take exactly one trigger and then stop taking more data so that the trace that was acquired can be inspected closely.

These needs are addressed by the radio button pair and three buttons in the data acquisition control pane.

The push button just below the radio button pairs is labeled **Start** when data taking is in active, clicking it begins data taking and changes the label and function of that button to **Stop**. The radio button pair determines exactly how data is taken. In **Auto** mode, if a trigger has not been processed within a timeout, a software trigger is generated and the resulting trace is displayed. If there is an accidental coincidence with a trace that is not making the trigger condition that can be used to determine the correct trigger position. Otherwise a baseline trace will be displayed. When the **Normal** radio button is selected, the automatic software trigger described above is not performed, and only the trigger conditions describe by the GUI will result in a trace.

If **Single Trigger** is clicked, data taking halts when the next trigger occurs. Clicking **Single Trigger** again accepts exactly one more trigger. If **Force** is clicked a single software trigger is performed.

If data taking is halted (by clicking **Stop**), the most recent traces remain displayed.

5.3 The Procedure Guide

The set of radio buttons at the bottom of the window describes the normal flow of setting up the digitizer. When a radio button is selected, only the controls relevant to that step of the process are enabled. This provides a wizard like interface for setting up the digitizer but in a way where all steps are visible at all times, steps can easily be skipped if appropriate and arbitrary backtracking is possible. If the user believes they know what they are doing the can, at any time select the **Expert** radio button and all controls will be enabled.

This set of guide modes is a compromise between the need to help novice users set up a digitizer and the impatience a full wizard interface would create in more expert or experienced users. Furthermore, laying out the full process to the user provides a quick reference guide for users who *think* they know what they are doing but get stuck doing it.

5.4 Interacting with the plot

The scope program provides support for several simple but useful interactions with the trace. These interactions can be performed either when data taking is active or when data taking is halted (e.g. after a single trigger or when data taking is halted). Less frequently used operations are menu commands while operations that we believe will be frequently performed are implemented using a select : operate model.

The position of the cursor is continuously displayed. This allows the cursor to be used to extract simple information about the trace such as the time and height of trace features.

If the cursor is dragged over the trace (MB1 with motion), a rectangular region of interest is created and displayed. The region is stacked below the trace so that the trace is always visible. As the region of interest is changed its extent in both time and signal height are continuously

displayed. When MB1 is released a pop up menu is displayed near the cursor.

- Doing nothing and eventually clicking **Cancel** provides a simple way to measure plot features. Most digital and analog scopes provide support for measuring voltage and time differences. This feature provides for simultaneous measurement of voltage and time differences.
- Clicking **Expand** expands the plot around the region of interest. Since Plotchart is allowed to choose the 'best' range and tick interval for a specified range of coordinate values, the actual expansion, in general will not be exactly the selected region of interest.
- Clicking **Info** provides the minimum and maximum values of the traces within the region of interest.

It is worth spending a bit more time describing the region of interest expansion. Multiple expansions are treated as a stack. Each expand operation pushes the previous plot limits to this expansion stack. The concept of an expansion stack was shamelessly stolen from BLT.

Whenver the scope display is expanded two buttons are displayed to the right of the guide pane. These buttons are labeled **Restore** and **Previous**. Clicking **Previous** pops the top of the zoom stack effectively going back one expansion level. Clicking **Restore** returns to the unexpanded display. Whenever the zoom stack is empty, these two buttons are removed from the user interface.

5.5 Menu commands

Less frequently used commands are relegated to the menu. The **View** menu provides the following functions:

- **Grid** toggles the visibility of a coordinate grid.
- **Real Coords** Toggles between axes labeled in ADC units (sample number and adc value) and physically meaningful coordinates (nanoseconds and millivolts).
- **Stab. Trigger** toggles on or off computations that stabilize the trigger position.

A few words about **Stab. Trigger.** There are two clocks in each module that run asynchronously. The sample clock is used to time the digitization of samples. The logic clock is used to clock the FPGA and other digital logic in the module. Triggers come a times that are related to the sampling clock but are determined by fpga firmware which is clocked by the logic clock. This means that left to itself, two completely identical signals may appear at different times in the traces. This can be distracting. Trigger stabilization is a software computation that locates the first trigger threshold crossing on the waveforms and shifts that to position it at exactly where it should be according to the value selected for the pretrigger. This stabilizes the position of the trigger in displayed time at the cost of moving the jitter to the points at the beginning and end of the waveform, which are normally uninteresting.

Turning on trigger stabilization while attempting to locate the baseline of the signal causes problems because traces acquired at that time may not have a discriminator crossing. Furthermore if the external trigger is used no meaningful shift can be computed because the discriminator may have had nothing to do with the trigger.

6. Packaging Concerns

From the very beginning, the goal was to package the software into a Starkit and bind that Starkit to platform dependent TclKits to form a set of Starpacks[7]. This would allow the application to be bound together with the CAEN Digitizer libraries providing for a nearly single file installation of the software on all supported

platforms (currently 32 and 64 bit Linux and 32 and 64 bit Windows operating systems).

Fully single file installation is not possible because operating system device drivers for the USB and CONET interfaces are required. These not only have to be installed but, in the case of Linux loaded into the kernel and, in the case of Windows placed where the hotplug system can locate and dynamically install them when devices are detected.

Getting cross-platform StarPacks to work with compiled extensions that have dependencies loaded in the virtual filesystem is tricky and, without a number of searches of **http://wiki.tcl.tk** this would have taken quite a bit longer to get working. The wiki provided quite a few hints about what the pkgIndex.tcl file might need to look like but was somewhat vague on how to get dependent libraries loaded from the StarPack itself.

The original idea was to provide a thin wrapper around **Tcl_FSLoadFile** that could load the Digitizer libraries and its dependencies as part of the process of loading the digitizer wrapper package. This posed no problems under Windows. Under Linux, however even though the documentation of that function states that if the object cannot be loaded from the virtual file system...” Tcl will attempt to copy the file to a temporary directory and load that temporary file”...the load consistently failed.

This was eventually traced to the fact that while the file may have been copied, evidently Tcl_FSLoadFile did not supply sufficient information about the location of the temporary copy to allow the dynamic loader to find it, so the dynamic loader fell back to searching for the shared object in its standard set of directories.

The eventual work-around for this problem was to wrap the StarPack in a modified shell archive. The modifications unpacked the Starpack into ~/.caenscope, creating the directory if needed, defined the environment variable **LD_LIBRARY_PATH** to include ~/.caenscope and executed the StarPack.

The pkgIndex.tcl script, detecting a Linux platform copies the appropriate shared libraries to that directory and explicitly loads them from there. Experimentally it appeared that simply specifying the full path to the copied shared libraries was still insufficient to load them without the **LD_LIBRARY_PATH** definition.

7. Future work

The functionality of the scope application is sufficient to support setup of the CAEN family of digitizer modules. Future work is planned both to extend the functionality of the software and to clean up some loose ends:

1. Support will be added to export traces from the display to both text and binary files. A simple XML or JSON format will probably be used for the text format and SQLite is being considered as a binary format that would support the random access retrieval of an event from a collection of events.
2. The architecture of the software mostly follows the Model View Controller (MVC) software pattern. The breaks in that pattern must be removed as there is discussion that the Model and Controller components should be able to run as a headless server with well defined socket based communications providing the capability of using either the CAEN provided View or a custom user written view that might drive the digitizers in a real data acquisition system.
3. The need to copy the read only SQLite module capabilities database out into the file system can be removed using work done by Anton Kovalenko [8] that I became aware of after implementing the database copy out code. Since the module

capability database is read only, this should be safe.

4. While I think the implementation of the device capabilities database as an SQLite database was a good choice, it leaves begs the question of how to maintain deployed versions of the scope program as CAEN expands our digitizer product line. Specific issues are:
   a) How to update the capabilities database.
   b) How to provide updates to the CAENDigitizer libraries when new hardware and even digitizer firmware updates require it.

8. Conclusions

With the increased use of high frequency, high resolution flash ADC based digitization cards, tools are required to support both inexperienced and experienced users of these devices. Tcl and StarPacks provide a development environment that is a good balance between quick development and, simple deployment.

This papers has described CAEN Technologies work to provide a tool for the initial setup of its own product line of digitizers. The software is moving towards its first initial public releases. For the most part development has been smooth and trouble free.

9. References

[1] CAEN Digitizer Library is a freely available support library for the CAEN family of digitizers available at:
http://www.caen.it/csite/CaenProd.jsp?idmod=717&parent=38

[2] Tcl++ is a C++ encapsulation of the major features of libtcl. It has been recently encapsulated as a .deb package and is briefly descdribed in
http://www.tcl.tk/community/tcl2004/Papers/RonFox/fox.pdf

[3] G. Howlett Building Applications with BLT
http://www.ing.iac.es/~docs/external/tcl/BLT/handouts.pdf

[4] B. Techentin et al.
http://rbctoolkit.sourceforge.net/

[5] Plotchart was written by Arjen Markus and is part of Tklib one manpage source is:
http://docs.activestate.com/activetcl/8.5/tklib/plotchart/plotchart.html

[6] SNIT is W. Duquette's pure Tcl object oriented extension to Tcl/Tk. SNIT is part of TclLib and is described e.g. at
http://www.wjduquette.com/snit/snit.html

[7] S. Landers *Beyond TclKit - Starkits, Starpacks and other stuff* Presented Tcl 2002 Vancouver BC Sep. 16-20, 2002 Paper available online at
http://www.tcl.tk/community/tcl2002/archive/Tcl2002papers/landers-tclkit/tclkit.pdf

[8] A. Kovalenko Tcl VFS integration for Sqlite3
http://www.siftsoft.com/tclsqlitevfs.html