Development of a Java-Based Application to Acquire and Analyze Oscilloscope Waveform Data

Paul Drake

Office of Science, Student Research Participant (SRP)

Argonne National Laboratory

Lemont, Il 60439

6 August 2010

Prepared under the direction of Edward May and Karen Byrum, High Energy Physics Division, Argonne National Laboratory

ABSTRACT:

A description of a Java program to facilitate data acquisition and analysis using a Tektronix TDS3000 series oscilloscope and applied to micro-channel photo-multiplier tubes is provided. A pulsed laser is aimed at two micro-channel plates in a light-tight box. The signals from these plates are fed into the oscilloscope, which displays the pulse shapes for each channel. Using an automated Java program, shell commands, and a C-program, the data from the oscilloscope is obtained, saved, analyzed and displayed to show a description of the pulse information. The Java program described here is a framework that could be amended to accept many different calculations and display options. For a description of the current laser and detector lab configuration, please see Eugene Yurtsev's paper "Effects of Transmission Line Readout," cited below.

INTRODUCTION:

The laser test beam for the Picosecond Timing Project has been utilized for several years with an array of traditional electronics hardware such as analog-to-digital converters (ADCs) and time-to-digital converters (TDCs) for data acquisition and analysis. To aid in this analysis and to expand the different types of analyses the lab is capable of making, an oscilloscope has been implemented to display the pulses from the Micro-Channel Plates (MCPs) as digital signals in time. To optimize the information obtained from the data provided by the scope, several shell scripts, a C-code, and a Java program have been created to obtain the binary data from the waveforms displayed on the scope, save and convert it to a comma-separated-value file of time vs. voltage, and begin to analyze the waveform for pertinent information.

HIGH LEVEL OVERVIEW

With a signal being fed from the MCPs to the oscilloscope, the java-based application acquires waveform data from one or more oscilloscope channels. The application allows the option of collecting data from a series of runs with one varied parameter, and is formatted to save the varied parameter in a data file with the rest of the waveform calculations. The data from the scope is collected in binary form and once it is collected it is converted to a .csv file and stored in a name-file hierarchy as follows:

-Main directory (specified by the user at startup)

-Parameter name (specified by user before each run)

-Ch1.isf, Ch1.csv, Ch2.isf, Ch2.csv, Ch3.isf, Ch3.csv, Ch4.isf, Ch4.csv

Once the data has been collected and stored, the Java application then analyzes each waveform to extract specific signal features for one or more pulses. This data is printed to the screen and stored immediately under the main directory as a space-separated .dat file for continued or later analysis. After each run, the program displays the selected waveforms using gnuplot, and at the end of the program displays a summary report of the signal features found by the java program.

METHOD: Data Acquisition

The oscilloscope that is utilized is a Tektronix TDS 3000 series 4-channel e-scope, which has a built-in Ethernet connection and assignable IP address. With the scope on the same network as the lab, a picture of the scope-image may be sent to a computer for viewing over an Internet browser. Further, using the "*wget*" shell command, the binary data from the scope image may be obtained in ordered pairs of time vs. voltage. A single channel is imported with each instance of "*wget*," and the channel number, outfile name, and outfile format (mathcad, spreadsheet, or internal) may be specified by the user. Since binary data is imported faster than mathcad or spreadsheet, a C-program has been provided to convert the binary data to a more traditional .csv file (provided by Yohie ENDO, obtained at http://yoheie.web.infoseek.co.jp/isftoasc/isftoasc.c). From here the waveform data may be graphed and displayed or analyzed by the Java program PulseWidth.java.

METHOD: Pulse Initialization

The Java class PulseWidth.class in the anlPulse/ package (and directory) provides a framework for waveform data manipulation for standard calculations such as pulse height, area, rise time, fall time, pulse width, and relative position in time. The program performs these calculations without fitting the data, so the calculations currently depend on good signal-to-noise ratio and how clean the signal is. Also, while the program can process up to two pulses for channel, the pulses must be clearly defined, with the signal falling below the threshold between pulses; however it would be possible to add a second threshold in code to provide a deliminator for the program to identify to separate the pulses.

Currently the user dictates the number of pulses to expect and the directory to which the data files will be saved. The program reads in the first 500 ordered pairs of the specified data file provided by the scope to be analyzed and averages them together to attain an approximation of the average noise. This assumes that a pulse does not occur within the first 500 data points. All 10,000 ordered pairs then are read in and assessed for the points where the signal reaches a relative

maximum[1]. These points are flagged as the maximums of the number of pulses specified, and pulse analysis occurs with these points as the defining points of the pulses.

METHOD: Pulse Description

PulseWidth.java creates a Pulse Object for each separate pulse in the data file. Once the maximum points of the pulse are identified, this information is passed to the Pulse class, where it creates a new instance of a Pulse. A threshold for the data file is determined from the noise average calculation, and the ordered pairs to either side of the maximum are compared to that threshold to determine the starting and ending points of the pulse. The maximum initially determined by the program is also redefined to ignore the average noise, so if the channel is offset from the origin the program still delivers a reasonable estimate of how much the pulse maximum differs from the channel's standard value. The x-value (relative time) for the starting, ending, and maximum of the pulse are stored in the pulse object for future reference. Defining the pulses in this manner presents a framework from which many different calculations may be made. The rest of the PulseWidth class demonstrates some of those calculations and provides a structure that allows for easy implementation of other such methods.

The PulseWidth class currently calculates the time scale of the data, start and stop, pulse width, pulse maximum, rise time, fall time, and area. Further, a method is provided for determining the derivative; however it is not currently implemented. The time scale is calculated by the difference between the 100^{th} x-value and the 99^{th} x-value, because there was some evidence of inconsistencies in time on either extreme of the data-file. Pulse width is simply determined by the difference between stop- and start- times. It should be noted that depending on the strength and shape of the signal, the threshold that is implemented in code as $2 * noiseAve$ may not be sufficient, which could lead to miscalculated start/stop times. Maximum is the relative maximum of the data minus the noise average.

Rise time and fall time implement an algorithm that calculates the ten-percent and ninety-percent points of the max and attempts to find approximately where in time they occur relative to each other. In the event that the ten-percent or ninety-percent mark lies between two adjacent data points, the program calculates the slope between them and finds the approximate corresponding time value from that using the equation $y = mx + b$. It should be noted that if the pulse is small in amplitude, or if there is a relatively significant amount of noise in the

1. Note: The program currently assumes that the pulses will be negative in amplitude, so the relative max of the data is displayed as a relative min.

channel, the ten-percent mark may not be accurately calculated in time and thus may lead to skewed results. This can also occur if the pulse-shape is not smooth, because the program may define the ten-percent mark at the wrong place in time.

To calculate the integral, the program uses the $x\text{-}values$ for the start and stop times, and every time increment in between them is multiplied by its corresponding $y\text{-}value$ and summed together. In essence, it is the equation $\sum_{i=start}^{stop} (x\text{-}increment)(y_i)$, which is a rough approximation of the integral.

Once these things are calculated for each pulse, they are logged in a gnuplot-ready file for simplified organization and further display and analysis. Gnuplot is a widely available tool for plotting and display of comma-separated-data files. All data from a series of runs is concatenated to the same .dat file, separated by comments that specify from which data file the calculations came. These files are always titled "dataCh#.dat," and stored directly under the main data directory specified by the user. Here the '#' represents the scope channel from which the data was collected. Note that the changing variable is always saved as a type String in the first column of the log file.

METHOD: AUTOMATION

In addition to these calculations, the Java program also enables automation of data acquisition and analysis. To do this, a main class is Scope.java is implemented. The main method of Scope.java initialized a class called Test, which in turn initializes the four separate classes of the package anlPulse: UI, Bash, PulseWidth, and GnupWrite. UI is simply a user-input class that handles I/O between the computer and user for argument handling. Bash serves as the intermediary between the Java program and the operating system for calling the shell script "wave4.sh" and gnuplot. "wave4.sh" (a bash shell script) obtains the binary data from the scope and converts it to comma-separated data, and the gnuplot call is used to immediately graph acquired data and to graph the analyses at the end of a series of runs. Bash.java interacts with the operating system through the java Runtime method, to which the command-line arguments are passed in order to communicate with the system. Note that the "wave4.sh" command is passed in the form of an array of strings, while the gnuplot arguments must be passed in the form of one single string. The Boolean "con" in the method processData(String[], Boolean) dictates which of the cases should be selected. The GnupWrite class manually writes out a tailored .dem (gnuplot commands) file to the chosen subdirectory to graph the waveforms immediately after receiving them to facilitate data comprehension. Each imported waveform is displayed twice, once as a

full trace and once with just a window around the pulses. Once a series of runs is completed, a new .dem file is written to the main data-storage directory that displays the results of the pulseWidth calculations. Five graphs are displayed: pulse height as a function of the changing variable, pulse height that is normalized to 1 relative to the first run, rise time, area, and area normalized to 1 relative to the first run. Note that there is an accuracy limit to gnuplot, and so for certain attempts to normalize data, gnuplot interprets the math as division by zero and will not display the graph.

After initializing these classes, Scope.java implements the UI class to prompt the user for a series of arguments: directory to save data, scope channels to import, number of pulses to expect, and the first of the changing variables. The Scope class creates the specified main directory as well as a subdirectory for the variable. For each pulse number to be imported, the save-directory, variable, and pulse number are provided as arguments to the Bash class, which calls “wave4.sh” to import data from that channel, convert it, and save it to the variable-specified subdirectory. Scope then calls the class PulseWidth with the arguments directory, variable, channels, and number of pulses to expect. For each channel, PulseWidth performs its calculations and saves them, along with the changing variable, in the top level of the directory specified.

METHOD: DISPLAY

Once all channels for a particular run have been imported and calculated, the Scope class calls GnupWrite to write the waveform .dem file (saved as “directory/var/waveFm.dem”) and then again to graph that .dem file with gnuplot. This produces a plot of the waveforms as they appear on the scope.

From here the user is prompted to enter another variable, which would start the data acquisition and processing over again. If at this prompt the user enters ‘q’ instead, the program will call GnupWrite again to write and graph a .dem file called “directory/finalPlot.dem,” which displays the five results from the PulseWidth calculations.

FURTHER IMPLEMENTATION

This program provides a solid framework for a more in-depth analysis of waveform data. Different fitting methods could be implemented, which could enable different calculations and interpretations or better approximations thereof. It would also be of use to provide an automated run option, where many runs could be made with one changing variable to produce a significant amount of data. Different options for the data display are also possible, such as histograms of a series of runs. This would allow for rms calculations or

possible Gaussian interpretations. For double pulses, the code could be altered to differentiate between overlapping pulses and perform the same calculations or to add individual pulses together to ensure correct operation of the micro-channel plates.

SAMPLE APPLICATION:

A series of runs were taken using the Tektronix TDS3000 series 300MHz oscilloscope with the 10μm transmission line (TL) MCP in channel "A" and the 10μm non-TL MCP in channel "B." The laser pulse-rate is varied from 1000Hz to 1000KHz. Oscilloscope channels are as follows: Ch1: A side (TL) top readout; Ch2: A side (TL) bottom readout; Ch3: B side (MCP). HVa = 2400 V, HVb = 2230 V with a light level of ~300 photo-electrons. Figure 1 shows a sample plot of the waveform plot displayed after each run with the java application. Displayed are the three waveforms for a pulser rate of 1000Hz. Figure 2 shows the feature-summary plot displayed at the end of the run. Note that with this ability the differences in rate-dependence may be seen automatically after taking the series of runs. The purpose of this specific study was to determine the rate of deterioration of the pulse as a function of rate. As the rate increases, more charge is extracted from the anode, and we believe that the effect in signal features is related to the size of the resistor change powering the micro-channel plate structure. This would imply that the TL has a significantly lower resistance in comparison to the non-TL MCP, which would prompt the pulse data to fall off faster in relation.

ACKNOWLEDGEMENTS:

I would like to thank Ed May and Gary Drake for their direct assistance with the data calculations. Thank you also to John Anderson for assistance with the e-scope setup, to Karen Byrum and to Jean-Francois Genat.

REFERENCES:

Arkin, Herbert and Raymond R. Colton. *Statistical Methods: As Aplied to Economics, Business, Psychology, Education, and Biology*. Barnes&Noble, Inc. New York, 1966.

TDS3000B Series Digital Phosphor Oscilloscopes: User Manual. Tektronix, Inc. <http://www.tektronix.com>

Yurtsev, Eugene. "Effects of Transmission Line Readout Electronics on the Timing and Spatial Resolution Properties

of Chevron Type Micro-Channel Plate Photomultiplier Tubes.” Department of Energy, SULI program. 1 Jun 2009.

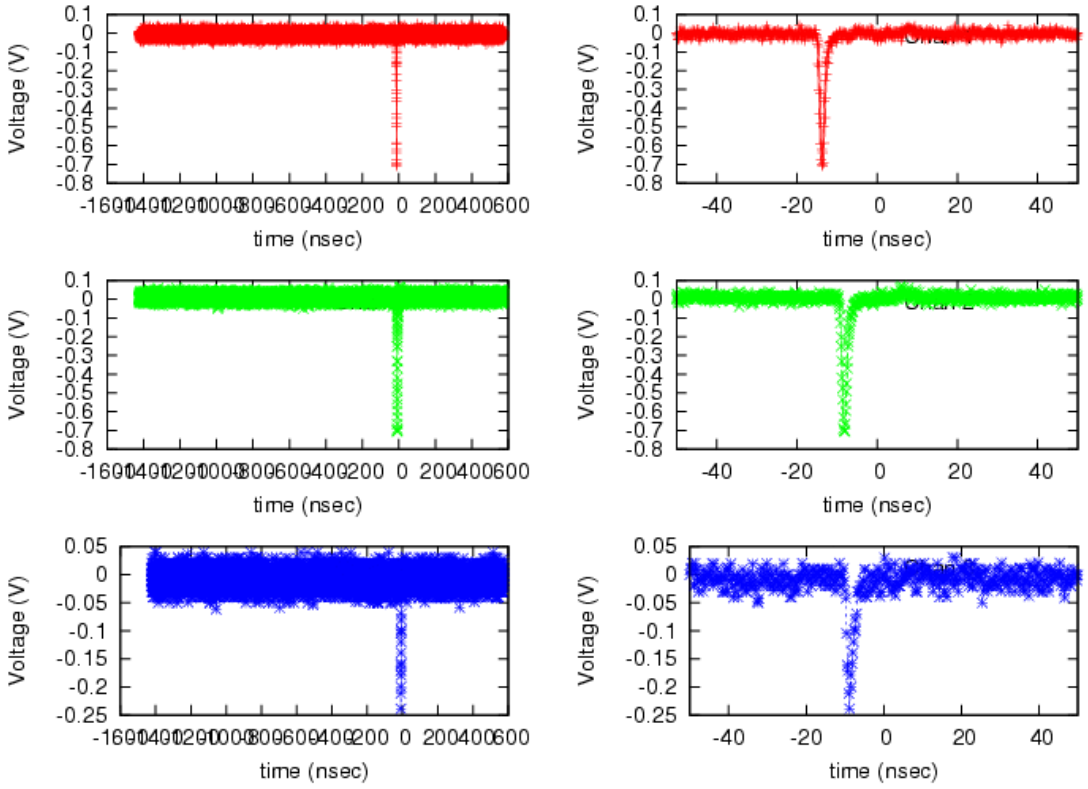

Fig.1: Sample of waveform raw data from channels 1, 2, and 3. Pulser rate at 1000Hz, light level at ~300phe

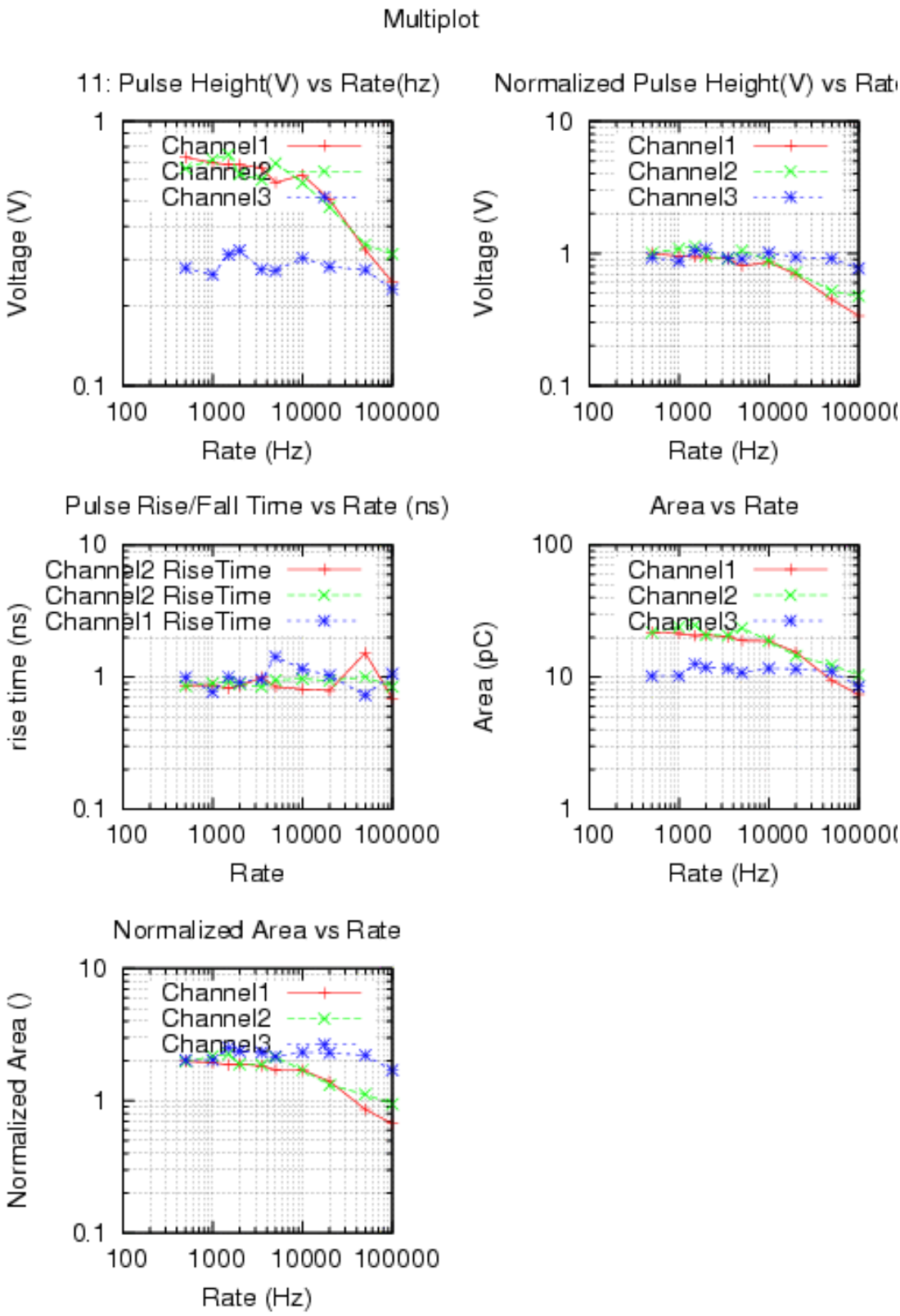

Fig 2: Data calculations from the java program. Note the difference in rate dependence between channels 1 and 2, which are the top- and

bottom- readouts of the TL, respectively, with the rate dependence in the non-TL MCP