Fast and Flexible Forth Programming in a Femtosecond Laser Lab

Theodore Sizer II

Laboratory for Laser Energetics
University of Rochester
250 East River Road
Rochester, New York 14623-1299

Abstract

A software system has been developed in Forth to manipulate, control, acquire, and display experiments which are performed in a femtosecond dye laser laboratory. Requirements placed on the system are that it be user friendly while maintaining both flexibility and speed. The use of vectored execution allows one to satisfy these two requirements while keeping the number of user addressable words to a minimum. This system is currently in use measuring both optical and electrical transients in the terahertz frequency regime. Techniques that are used to control and display the experiments will be explained along with some typical results that this system gives the experimenter.

Introduction

A software package has been developed which controls, manipulates, and displays experiments which are performed in a femtosecond laser lab. In the laboratory we produce pulses of light which are among the shortest ever produced in the world, and then use them as a strobe to "freeze" motion in biophysical, chemical, and semiconductor samples. The pulses of light are sixty-five femtoseconds in duration at a repetition rate of 1 kHz (a femtosecond is 10^{-15}th of a second or the distance that light travels in one femtosecond is .3 microns). With such a unique source to study fundamentals of physics, there are many proposed experiments each placing a different requirement on the data acquisition system. With each of these experiments comes a different experimenter so a new requirement must be added -- the ability for the system to be user friendly enough to be used by a large number of people while supporting the high repetition rate of the laser and the variety of different hardware used to do the experiments.

Forth, with its capability of high speed on a small machine, has been able to keep up with all experiments that have been done barring one done at 5 MHz (we are anxiously awaiting the time when Forth on a chip[1] could allow us to approach that speed). Vectored execution has been used throughout the code to allow the flexibility needed to support a great variety of experimental hardware. Two features have been implemented to make the system more user friendly: a file system for program development unencumbered with the confining nature of blocks, and further use of vectored execution to limit the number of words that the experimenter must know in order to run his/her experiment.

This system has been developed on a relatively inexpensive computer which illuminates another benefit of the system—the savings that one can obtain by using a small computer in Forth with intelligently designed software. With this system we have emulated the performance of a signal averager, boxcar integrator, lock-in amplifier, photon counter, and beam quality monitor at repetition rates up to 1 kHz. Additional benefits include the archival storage of data, user designed

display of the data for fast interpretation and the ability to control as well as monitor the experimental conditions. Without considering these important additional benefits the cost saving over purchasing each of these specialized devices separately is approximately 60%.

Hardware

The computer used in these experiments is a Digital Equipment Corp. LSI 11/23 with 256K of user addressable memory. There are four RS232 I/O ports for serial communication with terminals, experimental equipment, and a large mainframe computer for more extensive data manipulation in a multi-user environment. A CAMAC[2] crate is interfaced to the LSI backplane and is used for all data I/O other than the serial lines. CW and pulsed A/D's, a D/A, and several logic interface modules are used in the crate.

Two 8 inch single density disk drives are used for program and data storage—the software is written so that the programs are always in drive 0, and the data in drive 1 to avoid problems. An XY plotter is driven by three ports of the D/A for rudimentary plotting. Three stepper motor controllers have been interfaced through CAMAC, as well as an OMA[3] I used for one dimensional image analysis. The data is displayed on an ADM3A terminal with a Retrographics card which allows it to act as a Tektronics 4010 emulator. An auxiliary keyboard can simultaneously be used elsewhere in the lab through a different serial line. The video lines to the terminal's CRT have been tapped and manipulated into standard video so that the output can be viewed from anywhere in the lab on several different television monitors. The photograph in Figure 1 shows some of the hardware used.

Software

The Forth used is UR/LLE v. 3.01, a version which conforms roughly to the Forth-79 Standard[4]. Programming is done in a file system with text compression which frees one from the restrictions made due to blocks. This is particularly important for program clarity and the ability to add a plethora of comments and visually appealing space to the code without the worry of using excessive disk space. Details of this particular file system have been described elsewhere[5,6]. A data file management system has also been developed for archival storage. This system is separate from the programming file system and places a premium on economy of storage space with little overhead. A typical data disk holds 100 data runs of varying file lengths.

Software System Features

When using these ultra-short laser pulses as a strobe there are new experimental techniques that one must use to gather information about the sample under study. The technique most often used is called the "pump-probe" technique. As shown in Figure 2 the beam of light is divided in two by a beam splitter and the two beams travel separate paths until they recombine in the sample. In one of the paths we place a prism on a linear translator which is under computer control. By moving the prism back and forth one changes the length in space that the light must travel before it reaches the sample. Knowing the speed of light in air, we can calculate the time delay that this beam (the probe) has been delayed with respect to the other beam (the pump). Typical displacements are of the order of 1 millimeter which corresponds to 6 picoseconds (10^{-12} seconds) delay. By monitoring the change in the spectral intensity of the "probe" as a function of time delay from the "pump" we can obtain the response time of the sample after application of the "pump" pulse of light.

To perform this experiment the computer moves the stepper motor which drives the linear translator to the proper location, accumulates data, stores and displays the data, and then moves to the next location. A user designed display can lead to a closer interaction with the data. One consequence of this is that if an experiment is for some reason not performing properly the run can be aborted in mid-run saving valuable laser time. By averaging a number of laser shots at each translation stage position one can reduce the short term (shot to shot) noise and by taking multiple

scans of the entire length of the translator one eliminates long term drift. By appropriately balancing these two methods of data acquisition one can reduce nearly all sources of noise. The software can also be set to monitor the laser energy and reject a particular laser shot if it isn't within certain user defined bounds.

Figure 3 is an autocorrelation of the laser pulse used in the experiments. An autocorrelation is made by splitting the laser beam in two equal parts in a pump-probe setup, interfering them in a sample of potassium dihydrogen phosphate (KDP), and monitoring the frequency doubled light emitted as the translation stage is moved[7]. The pulse duration is 65 femtoseconds full width at half maximum. In order to take this run 2 scans with 40 points averaged at each point were accumulated taking approximately 2 minutes.

With two types of A/D's used and five different stepper motors to be controlled, one must be flexible in implementing the required A/D and stepper motor for it changes from experiment to experiment. Use of vectored execution makes this flexibility possible. Not only are the stepper motor and A/D drivers vectored but also the data manipulation that occurs during the run and the method used to display the data. Figure 4 illustrates the code used for the general data acquisition routine. By using this one general purpose routine and vectoring the different requirements into it, one not only saves memory space but forces the experimenter to remember only `DATA.ACQUIRE` to run his/her experiment after the initial setup. `REDO` is the main routine which runs the experiment without changing the experimental parameters from the previous run while `DATA.ACQUIRE` asks questions which determine the # of shots averaged etc. and then performs the run. `BEGIN-END` loops were used instead of `DO-LOOP`s so that the updated parameters evident in the code could be used in the installed programs.

Although the "step-average" method of data collection described earlier is used most often, there are times when one desires to acquire data in a manner similar to that of a signal averager. An experiment which can best use a signal averager type of operation is one which has low shot to shot noise but high long-term drift. In this method the stepper motor is set at one end of the scan range and then started towards the end of the scan. While it is moving, the data is sampled at a user-determined rate. By dividing the total travel by the number of data points obtained and by ensuring that the stepper travel is linear and continuous, one can later calculate the stepper position at each data point. This change in data collection scheme offers a clear example of the power and flexibility that one obtains when using a standard collection routine and vectored execution. Figure 5 illustrates the short programs that are written and then installed into the `DATA.ACQUIRE` routine. By entering the word `FAST.SCAN` one installs all of the proper programs into the execution vectors to change the operation of `DATA.ACQUIRE` from a "step-average" type of routine to the "accumulate while moving" type. The word `STEPPER.SCAN` returns you to the previous type.

The data that one obtains using this "fast scan" technique is shown in Figure 6. In this experiment the stepper motor was not controlling a linear translator but rather the wavelength control on a spectrometer to look at the different wavelengths of light emitted from a sodium atomic vapor sample[8]. The feature on the left is Rayleigh scattering of the laser, while the two on the right are the fluorescence from the two sodium D lines in the yellow part of the spectrum.

Although the execution vector used to move the stepper motor is called `MOVE.STEPPER` (singular) there is no reason that it can't move two steppers as well. The word `BEAM.PROFILE` installs the routines needed to move two orthogonal linear translators in a raster manner. Figure 7 illustrates the result in the form of the spatial beam profile of the laser beam obtained by raster scanning a photodiode with a 3 micron pinhole aperture across the laser which is focused by a 220 mm lens. The contour plot was made by porting the beam profile data to the large mainframe computer and using a resident contour plotting package.

The speed of the `DATA.ACQUIRE` routine allows for a repetition rate of 5 Hz with full screen updating of each laser shot or 40 Hz without screen updating. Frequently there is a need (usually for noise reasons) to accumulate data at a much higher rate. Machine code routines must then be written but can be installed into `ACQUIRE.DATA` in the existing `DATA.ACQUIRE` program. An

example of the data accumulated with the addition of the high speed code is shown in Figure 6 in a combination lock-in and boxcar integrator type experiment. An integrated circuit on a GaAs wafer was sampled with the probe beam which had its polarization rotated by the electric field strength in the circuit. The horizontal and vertical components of the polarization were monitored, A and B, and the voltage to the sample was turned on and off. The manipulated result that was averaged is then given by:

$(A-B)/(A+B)_{V\ on} - (A-B)/(A+B)_{V\ off}$.

To obtain the result 3.6 * 10^6 laser shots were taken. The measured risetime of the electrical waveform is 25 picoseconds[9,10]. By controlling the experiment (turning the voltage on and off) one emulates a lock-in amplifier – one only looks for changes which occur due to the changing voltage.

Many experiments which measure the amount of fluorescence from a sample after a laser has penetrated it are burdened by a very weak signal to examine. Much of the time the photomultipliers used to look at the sample see single photons of light striking them for each laser shot. A statistically superior method of dealing with these data is to actually count the number of photons that are created in a certain number of laser shots. Figures 9a & 9b are histograms of the amount of charge generated by a photomultiplier for a series of 3 * 10^4 laser shots. On the average there was less than one photoelectron generated in the photomultiplier tube for each shot. Figure 9a is the full histogram while Figure 9b is the same histogram with the zero charge component removed to see the one photoelectron spread in amount of charge[11]. A threshold value can then be determined to discriminate between a zero photoelectron event and a one photoelectron event. This method of accumulating data can be installed directly into the `DATA.ACQUIRE` routine as a different program in `ACQUIRE.DATA` or can be used on its own to monitor the output number of photoelectrons generated for each value of input laser energy obtained through a separate energy monitor.

As has been shown, using vectored execution in a general purpose routine allows one tremendous flexibility in choosing the type and speed of data acquisition. Variations in stepper motor hardware used and the way the data is manipulated also add to the user's ability to tailor the software to the particular requirements of the experiment. By appropriate choices of the installed routines and the use of a gated pulsed A/D, one can emulate the output of a signal averager, lock-in amplifier, boxcar averager, and photon counter.

Additional software has been written which, although it does not install into the `DATA.ACQUIRE` routine as the previous routines, does illustrate the extraordinary flexibility in data acquisition that Forth can provide.

Use of the large amount of free memory at high memory addresses allows one to make large two dimensional histograms. Many times there are experiments that investigate the variation in output energy with changing input energy. Since the output never has a one-to-one relation to the input but rather has a spread of different energies, the idea of a two dimensional histogram with the energy of the input on one axis and the energy of the output on the other is appealing. From this data one can calculate both the average output value and the standard deviation. Because Forth can accumulate data and increment a memory position in high memory fast, even a large 2-dim array can be filled in a reasonable amount of time. Machine code routines are installed into this program allowing the data acquisition speed to be increased to a rate of greater than 1 kHz.

Conclusions

As illustrated in the preceding example Forth can be used to design flexible, fast, and user friendly software for use in a laboratory. Creative use of vectored execution allow one the ability to "toggle" different experimental data acquisition techniques in and out while reducing the number of words that the experimenter needs to know. A major complaint regarding Forth, that it is too "wordy", can thereby be eliminated. Use of machine code routines at those points where speed is crucial and installing them into general high-level routines, repetition rates of greater than 1 kHz

have been obtained. Finally, it has been demonstrated that with a relatively inexpensive computer with software written in Forth one can emulate the operation of several widely used and expensive data acquisition packages while adding many additional features including data archival storage, experimental manipulation, and user designed displays.

Acknowledgements

The author is pleased to acknowledge the expert assistance of Steven Zoeller and Lawrence Forsley. Work supported by NSF grant no. PHY-8215132.

This work was partially supported by the Laser Fusion Feasibility Project at the Laboratory for Laser Energetics, which has the following sponsors: Empire State Electric Energy Research Corporation, General Electric Company, New York State Energy Research and Development Authority, Northeast Utilities Service Company, Southern California Edison Company, The Standard Oil Company (Ohio), and the University of Rochester.

Footnotes

1. Golden, J., Moore, C.H., Brodie, L. "Fast processor chip takes its instructions directly from Forth." *Electronic Design*, March 21, 1985.
2. Computer Automated Measurement and Control (CAMAC) is an industry standard for data I/O. Its operation is guided by IEEE Spec. # 583-1975.
3. OMA I is a registered trademark of Princeton Applied Research Corporation, a division of EG&G.
4. McCourt, Michael. *PDP-11 FORTH-79 Implementation Guide*, Computer Systems Group Technical Note #1. Laboratory for Laser Energetics, University of Rochester, May 1981.
5. This file system was developed with the sponsorship of The Institute for Applied Forth Research, Inc., Rochester, New York, 1982-1984.
6. Sizer, Theodore II, Forsley, Lawrence, and Helmers, Peter. "A File System in Forth: One Approach", *Proceedings of the 1983 Rochester Forth Applications Conference*, Institute for Applied Forth Research, Inc., Rochester, New York, 1983.
7. Sala, K.L., Kenney-Wallace, G.A., and Hall, G.E. "CW Autocorrelation Measurements of Picosecond Laser Pulses." *IEEE J. Quant. Elect.*, QE-16:990, 1980.
8. Sizer, Theodore II. "Picosecond Optical Collisions." Unpublished thesis. University of Rochester, Rochester, New York, 1985.
9. Valdmanis, J.A., Mourou, G., and Gabel, C.W. "Picosecond electro-optic sampling system." *Appl. Phy. Lett.*, 41/3:211, 1982.
10. Meyer, K. and Mourou, G.A. "Two Dimensional E-field Mapping with Subpicosecond Resolution." Paper from WB3-1 Picosecond Electronics and Optoelectronics Conference, Incline Village, Nevada, March 13-15, 1985.
11. The photomultiplier used in this experiment was an EMI type 9816B.

Manuscript Received July 1985.

Mr. Sizer received the B.A. degree in Physics and Astronomy from Amherst College, Amherst, Massachusetts in 1979 and the M.S. degree in Optics in 1980. He is working toward the Ph.D. degree with a concentration in picosecond optical collisions and ultrafast laser techniques at the University of Rochester's Institute of Optics.

Figures

Figure 1. Photograph of research associate Maurice Pessot adjusting the laser and the data acquisition hardware. From right to left: two patch panels (logic interfaces and input data lines) and disk drive, LSI 11/23 computer and CAMAC crate, Patch panel for different stepper motors and controllers, and large screen monitor to view the computer terminal display from the center of the room.

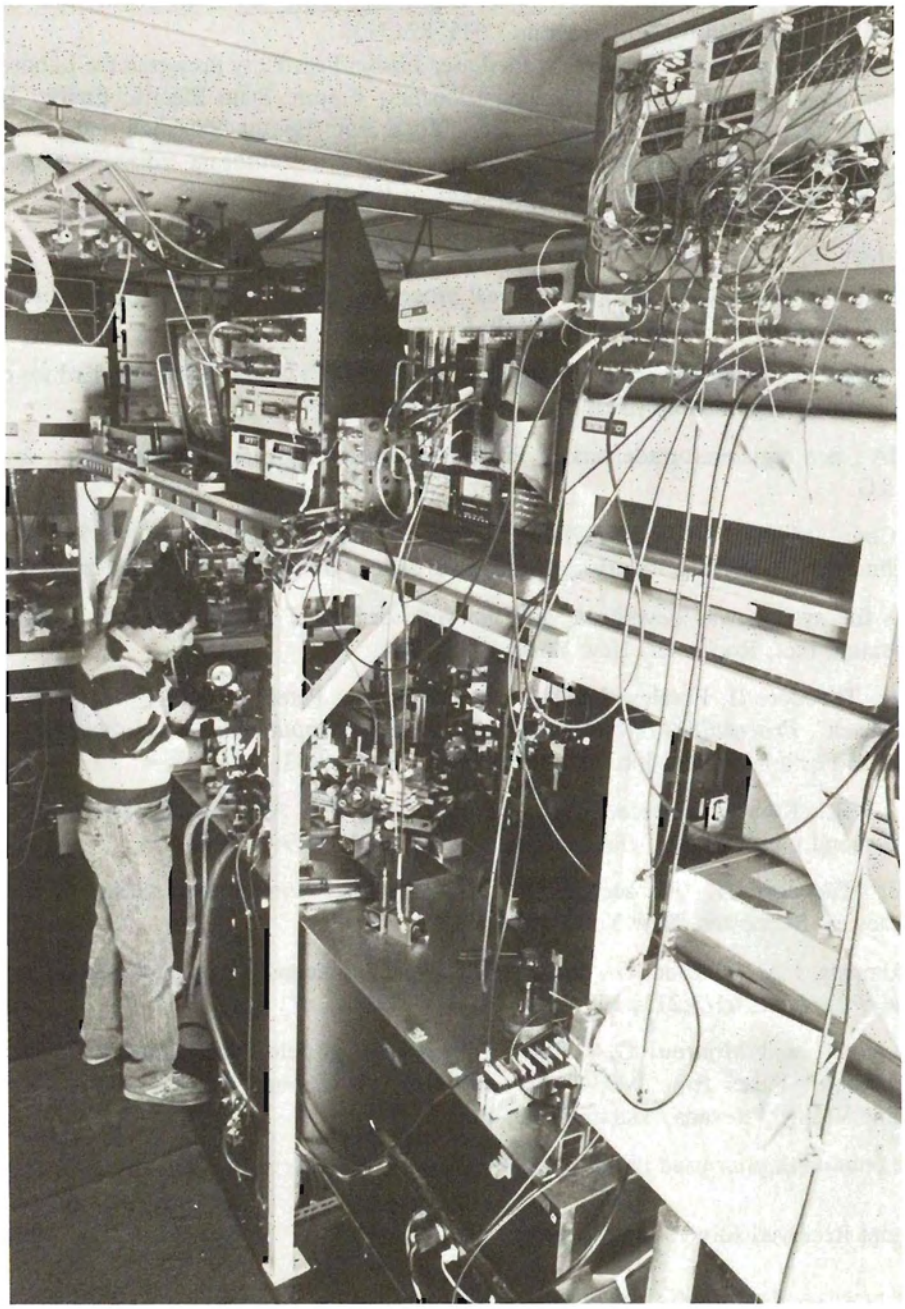

Figure 2. Pump-probe setup. An incident laser is divided into two paths traversing different paths and then focused together into the sample of interest. By changing the length in space of one of the paths one can change the time that the probe pulse is delayed with respect to the pump.

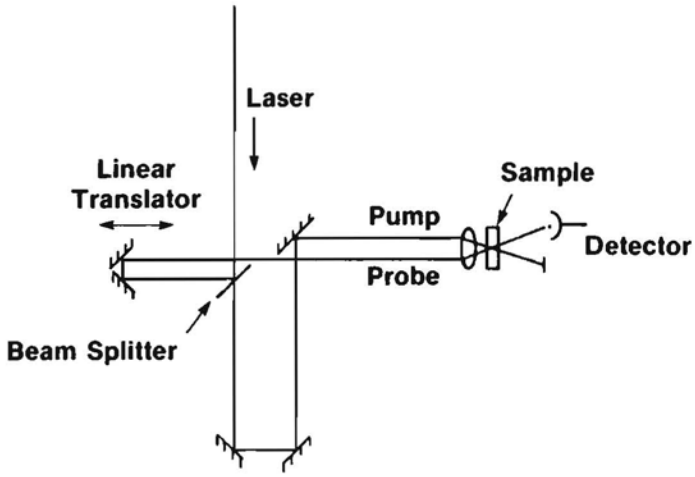

Figure 3. Amplified dye laser pulse. Pulse width is 65 femtoseconds FWHM assuming a hyperbolic secant pulse shape.

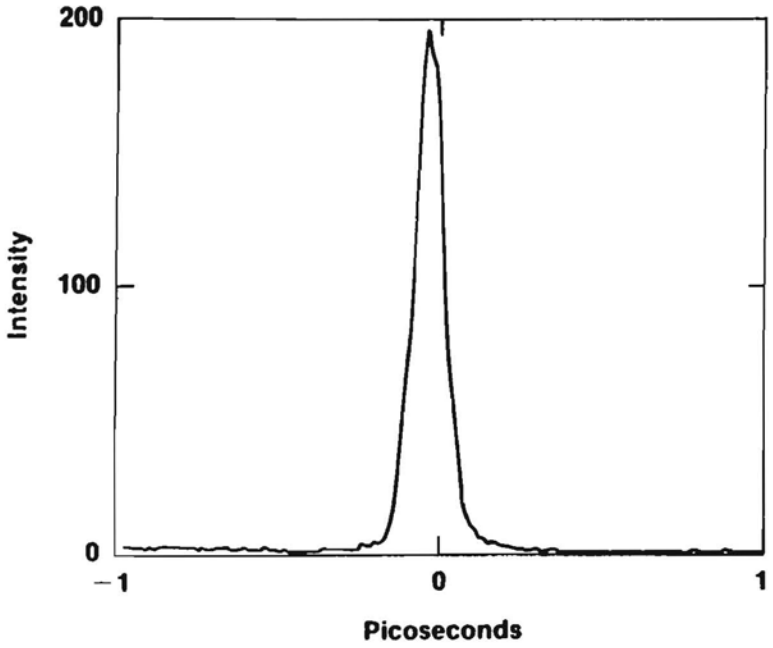

Figure 4. Forth code used for the general data acquisition routine.

The following programs are for data collection using the stepper motor to change experimental parameters, and to average and display the data during the experiment.

```
Ø XEQ ASKS.QUESTIONS      \ ask for input data
Ø XEQ INITIALIZE.DISPLAY \ setup the display
Ø XEQ MOVE.STEPPER        \ move the stepper motor in the program
Ø XEQ ACQUIRE.DATA        \ acquire the data
Ø XEQ MANIPULATE.DATA     \ manipulate the data
Ø XEQ DISPLAY.DATA        \ display the data
Ø XEQ STORE.THE.DATA      \ store the data, e.g. in an array
Ø XEQ START.STEPPER       \ start the stepper moving but don't wait
Ø XEQ STEPPER.IN.POSITION? \ returns 1 if in position, Ø if not
Ø XEQ FINISH.UP.THE.RUN   \ END THE RUN

Ø XEQ MV                  \ stepper move
Ø VAR EXECUTION.ADDRESSES  \ place to store current execution
18 ALLOT                  \ vector addresses

Ø VARIABLE #.PASSES       \ total number of passes
Ø VARIABLE PASS.#         \ current pass number

Ø VARIABLE #.POSITIONS    \ total number of stepper positions
Ø VARIABLE POSITION.#     \ current position number

Ø VARIABLE #.AVERAGE      \ total number to average
Ø VARIABLE AVERAGE.#      \ current average number

Ø VARIABLE #.STEPS        \ number of steps between positions

INSTALL #.POSITIONS IN #.DATA.PTS

1 VAR ?OFFSET             \ flag 1 if start scan at offset from Ø
                          \      Ø if no offset

: YES.OFFSET (  <>-<>, sets up to offset from zero at start )
   1 ?OFFSET ! ;

: NO.OFFSET ( <>-<>, sets up not to offset from zero at start )
   Ø ?OFFSET ! ;

Ø VAR ?DIRECTION          \ flag Ø if in + direction
                          \      1 if in - direction

: +DIRECTION ( <>-<>, sets the direction to be positive )
   Ø ?DIRECTION ! ;
```

```
: -DIRECTION ( <>-<>, sets the direction to be negative )
   1 ?DIRECTION ! ;

: REDO ( <>-<>, performs the same acquisition )
       (          as the previous DATA.ACQUIRE )

INSTALL #.POSITIONS IN #.DATA.PTS
                       \ use #.POSITIONS as the # of data pts
CLK.ON                 \ turn on the clock
INITIALIZE DISPLAY     \ setup terminal screen
Ø PASS.# !             \ reset the PASS.#
BEGIN                  \ start the pass loop
   PASS.# 1+!          \ add 1 to PASS.#
   Ø POSITION.# !      \ reset the POSITION.#
   BEGIN               \ start the postion loop
      POSITION.# 1+!   \ add one to POSITION.#
      MOVE.STEPPER     \ move to the next position
      Ø AVERAGE.# !    \ reset AVERAGE.#
      BEGIN            \ start the averaging loop
         AVERAGE.# 1+! \ add 1 to AVERAGE.#
         ACQUIRE.DATA  \ acquire the data
         MANIPULATE.DATA   \ manipulate the data
         DISPLAY.DATA      \ display it on the screen
         #.AVERAGE @ AVERAGE.# @ = \ check if at end of avg loop
      END                  \ if so, then exit
         STORE.THE.DATA       \ store it away
         #.POSITIONS @ POSITION.# @ \ check if at end of position loop
      = END                   \ if at end, exit
      #.PASSES @ PASS.# @     \ check if at end of pass loop
   = END                      \ if so, exit
   FINISH.UP.THE.RUN ;

: DATA.ACQUIRE ( <>-<>, asks questions and acquires data )

   ASKS.QUESTIONS             \ gets relevant parameters
   REDO ;                     \ acquires and displays the data

: DEFAULT.ASKS.QUESTIONS ( <>-<>, default program to ask questions )
   ADM CLS
   ." How many passes do you want?"
   3 OBT.DATA #.PASSES !
   ." How many positions do you want in a pass?"
   5 OBT.DATA #.POSITIONS !
   ." How many steps between positions?"
   5 OBT.DATA #.STEPS !
   ." How many points do you wish to average?"
   4 OBT.DATA #.AVERAGE ! ;

INSTALL DEFAULT.ASKS.QUESTIONS IN ASKS.QUESTIONS
```

```
: XEQ.ADDRESS.STORE       \ <>-<>, stores the current execution
                          \ vector addresses

   EXECUTION.ADDRESSES

    '  ASKS.QUESTIONS @        OVER !
    '  INITIALIZE.DISPLAY @    OVER 2 + !
    '  MOVE.STEPPER @          OVER 4 + !
    '  ACQUIRE.DATA @          OVER 6 + !
    '  MANIPULATE.DATA @       OVER 8 + !
    '  DISPLAY.DATA @          OVER 10 + !
    '  STORE.THE.DATA @        OVER 12 + !
    '  START.STEPPER @         OVER 14 + !
    '  STEPPER.IN.POSITION? @ OVER 16 + !
    '  MV @                    SWAP 18 + ! ;

: XEQ.ADDRESS.GET         \ <>-<>, retrieves the current execution
                          \ vector addresses

   EXECUTION.ADDRESSES

   DUP @       '  ASKS.QUESTIONS !
   DUP 2 + @   '  INITIALIZE.DISPLAY !
   DUP 4 + @   '  MOVE.STEPPER !
   DUP 6 + @   '  ACQUIRE.DATA !
   DUP 8 + @   '  MANIPULATE.DATA !
   DUP 10 + @  '  DISPLAY.DATA !
   DUP 12 + @  '  STORE.THE.DATA !
   DUP 14 + @  '  START.STEPPER !
   DUP 16 + @  '  STEPPER.IN.POSITION? !
       18 + @  '  MV ! ;
```

Figure 5. Simple instructions that are installed into the general data acquisition routine to change it to an "acquire while scanning" technique.

```
1000 ()DIM <FDATA>          \ array to hold the data temporarily
0 VAR #.OF.SCAN.PTS         \ # of points taken in one scan
0 VAR S.STEPPER.POSITION    \ starting position of the scan
0 VAR E.STEPPER.POSITION    \ ending position of the scan
20 VAR #.MSECS.TO.WAIT      \ # of milliseconds to wait between
                            \ each data point should be at least
                            \ 20 milliseconds because the cycle
                            \ time for the cw a/d is 18 ms

17 1SIGNAL @ GADR VARIABLE ADDRESS.OF.THE.DATA
                            \ address of the a/d in camac


: F.ASKS.QUESTIONS \ <>-<>, asks fast scan questions

   ADM CLS                  \ clear the screen
   17 1SIGNAL @ GADR ADDRESS.OF.THE.DATA !  \ reset the address location
                            \ so that a change of 1signal is noted
   ." How many passes do you want? "
   3 OBT.DATA #.PASSES !    \ acquire the # of passes

   ." Starting position to scan from ? "
   6 OBT.DATA DUP S.STEPPER.POSITION !
                            \ acquire the starting position

   ." Ending position to scan to ? "
   6 OBT.DATA DUP E.STEPPER.POSITION !
                            \ acquire the end position

    > ?DIRECTION ! ;        \ set the direction flag for graphics


: F.MOVE.STEPPER \ <>-<>, sets up the stepper and moves it

   0 #.AVERAGE !            \ the number to average is always 0
   1 #.POSITIONS !          \ set # of positions to 1
   S.STEPPER.POSITION @     \ move to the starting position
   TAKE.OUT.BACKLASH MV     \ while taking out backlash
   E.STEPPER.POSITION @     \ start the stepper towards the end
   START.STEPPER ;          \ but don't wait for an in position flag
```

```
: F.ACQUIRE.DATA \ <>-<>, acquire the data

   ADDRESS.OF.THE.DATA @@  \ fetches the data from the camac address
   1SIGNAL @ PED# @ -      \ subtract the pedestal value away
   AVERAGE.# @ 1- <FDATA> ! ; \ store data in <fdata> temp array


: F.MANIPULATE.DATA \ <>-<>, check to see if at the end of scan
                    \ or have filled up available memory
   #.MSECS.TO.WAIT @ MSEC  \ wait the prescribed # of msecs
   1000 AVERAGE.# @ = IF   \ if # of data pts taken is >1000
      CR ADM CR CLS ." Memory out of range !!!!!!! "
      RESTART              \ then abort
   THEN
   STEPPER.IN.POSITION?    \ is the position flag set ?
   IF AVERAGE.# @ #.AVERAGE ! THEN ;
                        \ if so, exit the data acquisition loop
```

```
: F.STORE.DATA \ <>-<>, transfer the data to main array <data
               \  taking into account the # of scans taken
   S.STEPPER.POSITION @    \ start the stepper back towards the
                           \ starting loc again for next scan
   ?BACKLASH @ IF          \ if desired, take out backlash
      AMT.OF.BACKLASH @
      ?DIRECTION @ NOT IF NEGATE THEN
      +
   THEN START.STEPPER      \ send stepper to start loc but do
                           \ not wait for in position flag
   PASS.# @ 1 = IF         \ if this is the first pass, then
      CLG 20 MSEC CLS      \ clear the screen
      #.AVERAGE @ 0 DO     \ loop through all of the data pts
         I E.STEPPER.POSITION @ S.STEPPER.POSITION @ -
         #.AVERAGE @ */ S.STEPPER.POSITION @ +
                           \ calculate the x position that the
                           \ stepper was in when the data was
                             taken
         I 2* <DATA !      \ store this into the x location
         I <FDATA> @ I 2* 1+ <DATA !
                           \ store the y value into y loc
      LOOP
      #.AVERAGE @ #.DATA.PTS ! \ set the # of data pts = to the
                               \  # of data pts taken
      GRAPH                     \ graph out the data

   ELSE                         \ if this is not the first scan
      #.AVERAGE @ #.DATA.PTS @ MIN 0 DO
                               \ then loop through the data pts
                               \ that were retaken on this scan
         I 2* <DATA @          \ fetch the x data from prev scans
         I 2* 1+ <DATA @ 2DUP \ fetch the y data from prev scans
         ERASE.PT *DRAW        \ erase the point on the screen
         S->D PASS.# @ 1- D*   \ mult the y value from
                               \ prev scans by the # of prev scans
         I <FDATA> @ S->D D+   \ add this scan to it
         PASS.# @ 0 DP/MOD     \ and divide by the new total of scans
         2SWAP 2DROP DROP DUP
         I 2* 1+ <DATA !  \ store the averaged value into <data
         POINT.PLOT *DRAW  \ and plot the point out
      LOOP
   THEN
   BEGIN
      STEPPER.IN.POSITION? \ wait until stepper has returned
   END ;                   \ to the starting position
```

```
: STEPPER.SCAN \ <>-<>, resets the parameters to the slow
                     \ stepper scan mode

        ASKS.QUESTIONS @  '  DEFAULT.ASKS.QUESTIONS 2 - <> IF
                          \ if the data acquisition is not set for
                          \ the default acquisition mode, then
     XEQ.ADDRESS.GET      \ fetch the addresses of the installed
                          \ routines in DATA.ACQUIRE
     INSTALL #.POSITIONS IN #.DATA.PTS \ and return the # of
                          \ data pts to # of positions
   THEN ;


: FAST.SCAN \ <>-<>, sets the parameters to do a fast scan

        ASKS.QUESTIONS @  '  F.ASKS.QUESTIONS 2 - <> IF
                         \ if the data acquisition is not set for
                          \ fast acquisition mode then
        XEQ.ADDRESS.STORE \ store away the current execution
                          \ vector addresses
        INSTALL #.OF.SCAN.PTS IN #.DATA.PTS
                          \ set the # of data pts to # of
                          \ scan pts
        INSTALL F.ASKS.QUESTIONS IN ASKS.QUESTIONS
                          \ reset questions
        INSTALL F.MOVE.STEPPER IN MOVE.STEPPER
                          \ reset stepper commands
        INSTALL DUMMY IN INITIALIZE.STEPPER
                          \ don't init stepper
        INSTALL F.ACQUIRE.DATA IN ACQUIRE.DATA
                          \ change data acquisition
        INSTALL F.MANIPULATE.DATA IN MANIPULATE.DATA
                          \ manipulate data for scan mode
        INSTALL DUMMY IN DISPLAY.DATA \ display taken care of
                          \ in manipulate data
        INSTALL F.STORE.DATA IN STORE.THE.DATA
                          \ new store data routine
     THEN ;
```

Figure 6. Fluorescence from the Na D lines and laser Rayleigh scattering.

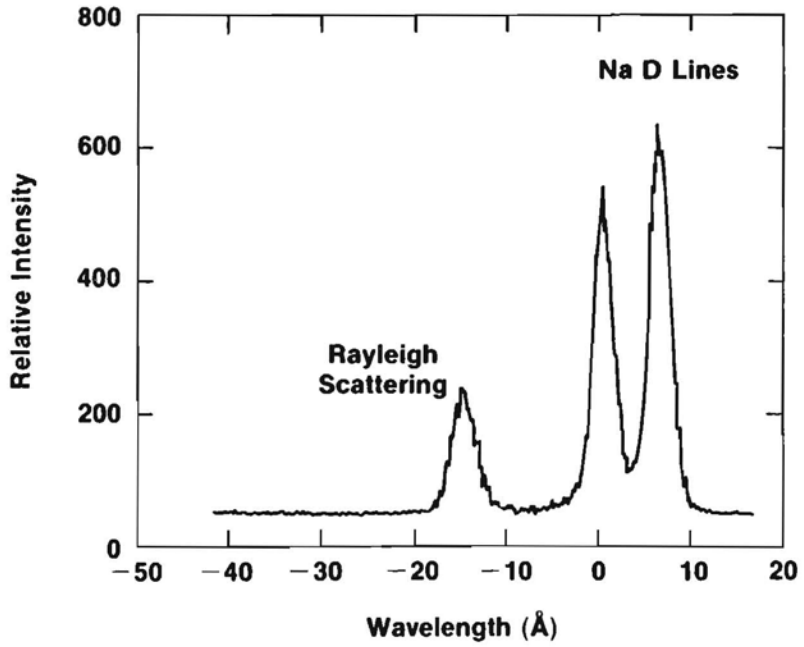

Figure 7. Spatial laser beam profile at the focus of a 220 mm lens.

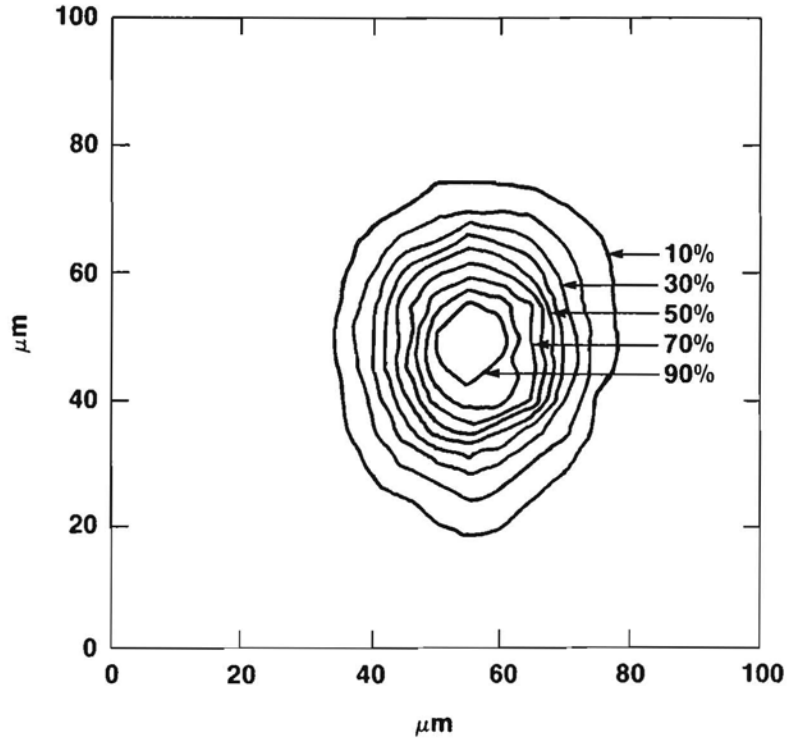

Figure 8. Electrical risetime of a pulse on a GaAs circuit sampled using the +hybrid boxcar-lock-in type of data acquisition.

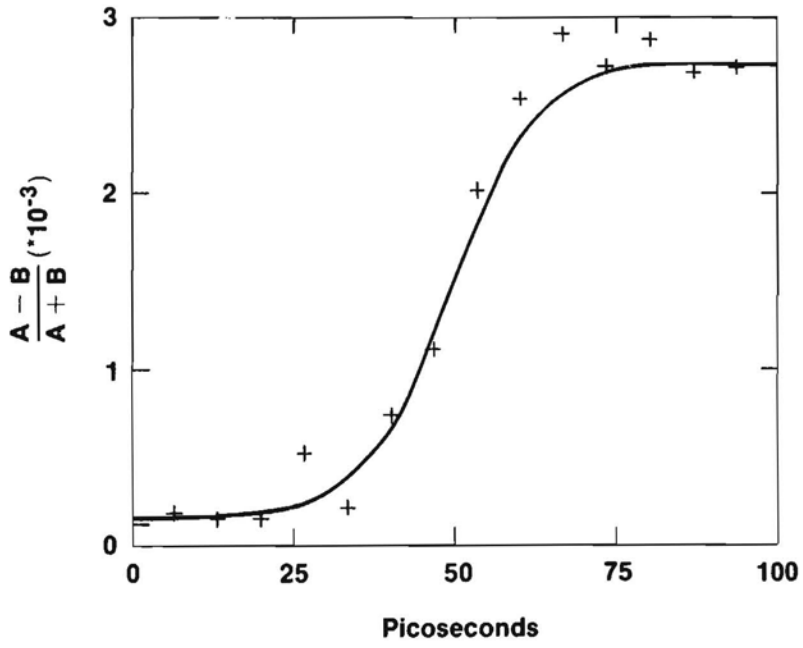

Figure 9a. Histogram of 30000 laser shots versus charge generated by the photomultiplier tube.

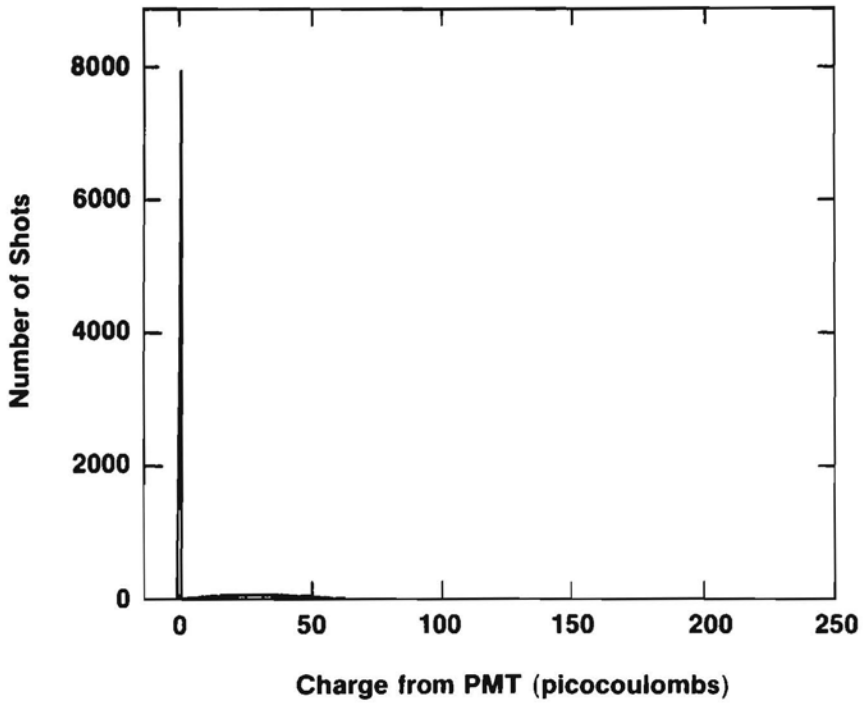

Figure 9b. Same as 9a with the zero photoelectron components removed so that the one photoelectron components are visible.