4 MPScope 2.0

A Computer System for Two-Photon Laser Scanning Microscopy with Concurrent Plasma-Mediated Ablation and Electrophysiology

Quoc-Thang Nguyen, Jonathan Driscoll, Earl M. Dolnick, and David Kleinfeld

CONTENTS

4.1 INTRODUCTION

The MPScope[1] freeware package is a comprehensive suite of programs that run under the Microsoft Windows operating system and are designed as control and analysis elements for two-photon laser scanning microscopy (TPLSM) and various extensions. MPScope consists of MPScan, the acquisition software, MPView, the analysis program, and MPFile, a utility to open MPScope data files in user-written applications. MPScope was designed from the outset as a turn-key, stand-alone, easy-to-use software system with flexible hardware options aimed at concurrent functional imaging, (e.g., calcium fluorescence), and electrophysiology experiments. However, the versatility of MPScope is such that it can be used in many different types of experiments including in vivo cortical blood flow imaging and two-photon plasma-mediated ablation automated histology.[2] The usefulness of MPScope stems mostly from leveraging advanced software technologies available on the Windows platform.

Over 50 laboratories have registered for either using MPScope or for downloading its source code as of summer 2008. Many users run MPScope on custom-made two-photon laser scanning microscopes designed for in vivo imaging, such as the one described by Tsai and Kleinfeld[3] in Chapter 3, while others have successfully adapted MPScope to control multiphoton systems based on commercial microscopes. The widening use of MPScope had, unfortunately, the unforeseen side effect of spurring a multitude of disparate hardware configurations, some very specific to a given laboratory, with the concomitant burden of supporting them in successive releases of MPScope. To alleviate this situation for future users of MPScope, we have overhauled MPScope and formed a presumably stable baseline configuration.

4.1.1 An Overview of Software for Two-Photon Laser Scanning Microscopy

In every laser scanning microscope, luminance signals from discrete locations (i.e., pixels) are combined to form an image registered with the spatial excitation pattern of the laser beam. Synchronization of excitation signals, digitization of emitted light, and subsequent frame reconstruction and processing is best done by software. Computer programs that control two-photon laser scanning microscopes are by no means different from those used in conventional laser confocal microscopes. In both types of imaging, the frame acquisition programs are tightly related to the hardware design of the microscope, while analysis applications are much more flexible, thanks to the existence of widely used standard formats for imaging data. The challenge in developing a useful software package for in vivo TPLSM is to cover a wide range of possible experiments with a large choice of auxiliary hardware without unduly complicating the user interface and limiting the performance of the microscope. To avoid the time-consuming task of writing a two-photon microscope program *de novo*, users of two-photon microscopes based on modified Nikon Fluoview confocal platforms can take advantage of the original commercial software by extending it with custom add-ons.[4] For custom-made microscopes that use resonant scanner mirrors (e.g., the CRS series by GSI Lumonics), programs like e-Maging[5] are capable

of displaying and streaming to disk up to 30 frames per second with each frame comprising 500 by 450 pixels.

4.1.2 Earlier Matlab™- and LabVIEW™-Based Two-Photon Microscope Software

Most current two-photon microscope systems, such as the one described in this book,[3] use a pair of galvanometric mirrors that are arranged orthogonally to generate the laser beam scanning pattern in the X-Y plane. Although slower than microscopes that use resonant scanners, these systems are more flexible as they provide finer control of the pixel dwell time, allow rotating full frames or line scans, and give the possibility to scan arbitrary regions of interest or to park the laser beam at a particular spot.[6] These microscopes, for example, can be controlled by ScanImage,[7] a software developed by the Svoboda Laboratory and written in the Matlab™ programming language (MathWorks, Natick, Massachusetts). Prior to MPScope, our laboratory and that of several affiliates ran a program developed in-house that was written in the graphical language LabVIEW™ (National Instruments, Austin, Texas).[8]

4.1.3 Drawbacks of Matlab™ and LabVIEW™ Development

The popular programming environments of Matlab™ and LabVIEW™ rely on some form of interpretation of the source code using a proprietary run-time engine. This approach provides, at first, a shorter development time for small- to medium-size projects, especially since many two-photon microscope users are already proficient with Matlab™ or LabView™. However, the interpreted nature of these programs imparts a significant penalty in performance and computer resource footprint. The dependence on version-bound run-time libraries makes the code particularly fragile to upgrades of the development environment. Severe backward compatibility issues occurred when Matlab™ went from versions 6.5 to 2007 and when LabVIEW™ v.6 was superseded by v.7. Perhaps the biggest drawback of developing in LabView™ or Matlab™ is that they are not designed for large and complex applications, unlike general-purpose languages such as Microsoft Visual Basic or C/C++. Although Matlab™ is essentially aimed at scientific calculations using floating-point arithmetic, a laser scanning microscope program spends most of its time dealing with integer data. Conversely, LabView™ is obviously optimized to construct data acquisition programs but is limited in its algorithmic capabilities. For both programming environments, some of these drawbacks can be alleviated by writing critical tasks in a compiled language such as C/C++, packaging these functions into Dynamic Link Libraries and calling them from the main program. This technique has the drawback of having to maintain two different code bases, with the concomitant problem of deploying a Dynamic Link Library always in synchrony with the current version of the main program.

4.1.4 MPScope 1.0: Technical Choices

The incentive to write MPScope was based on two considerations:

1. Although computer programs already exist to control two-photon microscopes, there is a considerable interest in increasing the frame rate and the on-line image processing. Both ScanImage and our past LabVIEW™ program use an asymmetrical, sawtooth-like scan pattern for the fast X mirror. A more efficient scanning scheme is to use a symmetrical waveform that allows acquisition of another scan line on the return path of the mirror.[5] Such a scan pattern increases the overall frame rate, potentially reduces photobleaching by using the retrace time to collect pixels, and has the added advantage of driving the mirrors with a waveform that contains less high frequencies than an asymmetrical waveform. From a user standpoint, it is highly desirable that the program display images with lines in the correct order while acquisition takes place. This requirement implies that the software should invert each alternate line, sampled in reverse order, in near real-time; misalignment results in comb-like features around objects viewed in full frames. Because other computer-intensive tasks have to be performed simultaneously on the incoming data, e.g., calculating and plotting intensities of region of interest, on-line fluorescent-energy resonant transfer calculations, etc., we felt that neither LabVIEW™ nor Matlab™ interpreted programs would have the required speed and future growth capabilities for fast full-frame two-photon imaging.
2. Two-photon imaging experiments are becoming more complex. In many experiments, basic functions of two-photon microscopy programs need to be successively executed automatically. A simple task, such as the acquisition of a 3D volume by repeatedly taking a stack and moving the field of view over a large preparation, is an example where customization and automation of microscope function can reduce the user workload. Inclusion of a feature to easily customize and automate tasks was a priority when developing MPScope.

4.1.4.1 Hardware Principles

The basic task of a laser scanning microscope is to output the coordinates of the laser beam position while acquiring the emitted fluorescence of the corresponding pixel during a dwell time that lasts typically from 1 to 10 μs. At these rates, programmed input/output operations and interrupt-driven input/output routines are too slow to achieve the necessary throughput; for a benchmark, see Reference 11. On the Windows platform, dispatching an interrupt takes at least a hundred instructions between the activation of the Interrupt Request Line and the servicing of the interrupt.[12] Moreover, since these techniques rely on the processor for their execution, the more they are called the less time remains for the processor to display and store the incoming pixels.

The above input/output issues are largely circumvented through the use of Application Specific Integrated Circuits located on modern data acquisition boards. For instance, the Mini-MITE circuit on the National Instrument PCI-6110 S-series board used by MPScan controls Direct Memory Access (DMA).[13] Here, the Mini-MITE takes ownership of the Peripheral Connect Interface (PCI) bus and thus

becomes a "busmaster" to transfer data from/to main memory to/from the periphery without intervention of the processor. The Mini-MITE allows also several input–output-independent DMA operations (e.g., analog/digital, digital to analog, and digital input/output) to be performed simultaneously via separate paths called channels. Moreover, the Mini-MITE circuit can manage data transfer from/to buffers that appear continuous in the program memory space but are fragmented in random access memory due to virtual memory paging. This capability, called "scatter-gather DMA," allows arbitrarily large buffers in random access memory to be transferred from/to periphery.

The data transit to digital-to-analog converters or from analog-to-digital converters occurs via small first in/first out (FIFO) memory buffers that are located on the data acquisition board. The FIFOs are filled or emptied in small bursts of bus activity, usually when the FIFOs reach a half-empty state. All these complex operations are initiated, controlled, and terminated by data acquisition device drivers. In addition, the drivers are also responsible for the critical page-locking of scatter-gather DMA buffers, which prevents their swapping from main memory to disk during data transfer. In the case of National Instruments hardware, data acquisition device drivers are managed by the NI-DAQ™ or NI-DAQmx™ software layers that hide the complexity of register-level hardware programming by providing a uniform, device-independent programming interface to applications.

MPScan relies on large memory buffers to store several most recently acquired images and the entire scan pattern for a single frame (Figure 4.1). The amount of memory allocated at the application level is doubled at the device driver level. This memory, allocated by NI-DAQ™, is organized as a circular buffer to perform double-buffering DMA operations in auto-initialize mode.[14] During frame acquisition, when half of the DMA buffer zone has been filled, NI-DAQ™ copies the recently acquired frames from the DMA buffers to the MPScope buffers while acquisition takes place in the other half of the DMA buffer. NI-DAQ™ then alerts MPScan of the availability of new frames. The size of the buffers is optimized to provide enough time for MPScope to process up to one second of incoming frame data, an interval with a large safety factor, without incurring an overrun of data in the DMA buffers. A similar scheme is implemented in reverse to insure that the laser beam X and Y position data for the digital to analog converter are placed in a large buffer that contains all of the pixel coordinates of a frame. This arrangement also allows immediate updating of the X-Y pattern when the frame needs to be rotated.

Timing synchronization between the digital-analog and analog-digital conversions is ensured through the use of the same pixel clock for the two concurrent operations and by triggering on common internal signals. The routing of the clock and trigger signals is performed by the digital acquisition system timing controller, an application specific integrated circuit that is located on the data acquisition board.[15] By using the full capability offered by such application-specific integrated circuits, only one multifunction board is required to output the scan pattern while simultaneously sampling the inputs.

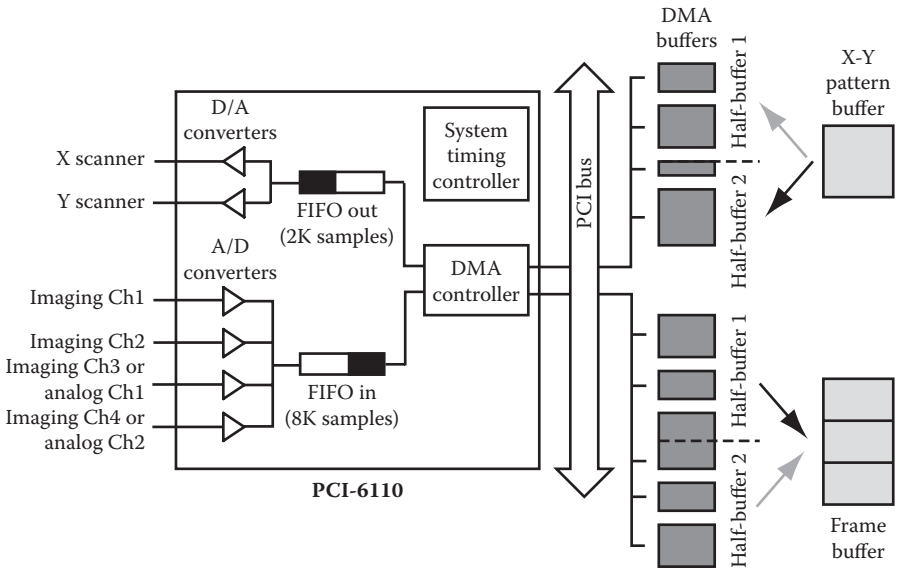


FIGURE 4.1 Data flow for MPScan acquisition and scanning. With Intel processors, DMA buffers can be fragmented into smaller, discontinuous memory zones composed of 4-kB pages of physical memory. Prior to a DMA transfer, the NI-DAQ™ driver page-locks the DMA buffers and programs the DMA controller (Mini-MITE), an application specific integrated circuit, with a linked list of memory zone descriptors. During scatter-gather operations the DMA controller goes through the list to determine the physical address of the memory locations to perform the DMA. The system timing controller is an application specific integrated circuit that synchronizes transfer operations.

4.1.4.2 Extensive Multithreading

For continuous, gap-free acquisition, the NI-DAQ™ and NI-DAQmx™ libraries enforce a cyclical data processing scheme whereby the application alternates between fetching frames from large buffers and processing the frame data. However, to enhance the responsiveness of the acquisition program, it is preferable to uncouple these two activities by splitting them into separate threads. The core of MPScan consists of a data pumping thread organized as an infinite loop that waits for the device driver to fill its frame buffer. Once this operation is performed, the data pumping thread reformats the data by splitting the sample stream into separate channels, reorders pixels in alternate lines, discards pixels at line turnarounds, reorders lines in alternate frames, saves frames to disk, and dispatches the frame data to client threads. These client threads include the frame display, the real-time frame averaging window, the real-time pixel distribution graph, the window that displays average intensity from regions of interest, and the analog data oscilloscope screen.

Care was taken to ensure proper load balancing between threads and to avoid producer/consumer deadlock problems using Windows synchronization primitives. Client threads have their own set of frame buffers that can be written by the data-pumping thread when authorized by a guarding semaphore or event. This allows

client threads to process their data without interference from the data pumping thread, or to have them go to sleep in a very efficient way for the central processor unit, while waiting for the data pump thread to signal the availability of new frames. Other threads created by MPScan include the one that communicates with particular X-Y-Z stage controllers. The communication thread method allows background processing of acknowledgment replies sent by the controller, which can occur, especially in the case of a long displacement, several seconds after MPScan sends a command. The creation of a communication thread thus prevents the entire program from hanging up after commands to the X-Y-Z stage are issued.

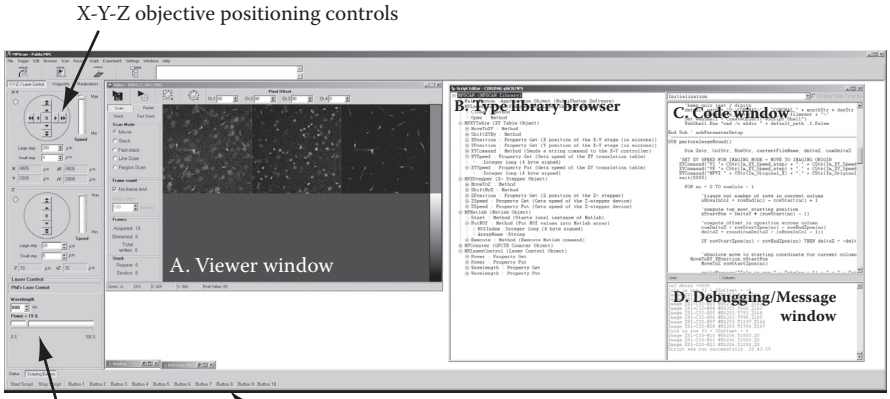
The multithreaded architecture of MPScan is ideally matched for current multiprocessor and multicore computers. This computing power allows several tasks to be performed entirely in software that, in single processor computers, were performed by specialized hardware with embedded central processor units or digital signal processors. This includes rotation of the scan directions and various timer/stimulators. A problem that multithreading does not solve, however, is the generation of highly reproducible timing intervals. The most accurate timer under Windows is the multimedia timer that allows, at best, only a 1 ms resolution with possible unpredictable delays. The time resolution and accuracy of the multimedia timer are insufficient to provide sub-millisecond, high-precision intervals that are critical to stimulate live preparations. For these and related tasks, additional PCI hardware featuring onboard timers is necessary. Finally, the large memory capacity associated with current computers, i.e., at least 8 Gbytes of random access memory, is adequate enough to store vast numbers of successive frames simultaneously in memory.

4.1.4.3 Object-Oriented, Native-Compiled Code

The choice of an object-oriented native code compiler to develop MPScope, as opposed to compilers outputting interpreted byte codes such as .Net compilers or Java, offers several advantages over Matlab™ and LabVIEW™. Speed is the primary issue, since repetitive processing has to be performed in the data pumping thread in order to rearrange the incoming pixel stream into frames, especially since MPScan outputs symmetric line scans and thus has limited fly-back time.

We developed MPScope in the Object Pascal language with the Delphi™ Integrated Development Environment (CodeGear, Scotts Valley, California). Delphi™, a modern descendent of Turbo Pascal, has a particularly short compilation time. This feature was especially appreciated during the development of MPScan, which required frequent trial-and-error runs to adjust NI-DAQ™ function call parameters in order to obtain the desired data acquisition behavior. Delphi™ comes with an object-oriented framework of software components called the Visual Component Library that permits easy development of Windows applications, while still allowing complete access to the Windows Application Programming Interface. One advantage of using a framework that relies on native Windows controls, unlike LabVIEW™, for instance, is the lack of constraints on the "look and feel" of the application. For concreteness, we show the MPScan user interface during a two-photon automated histology session in Figure 4.2.

The use of native Windows Application Programming Interfaces is also critical as a means to optimize bus-intensive graphical operations that handle device-independent



E.

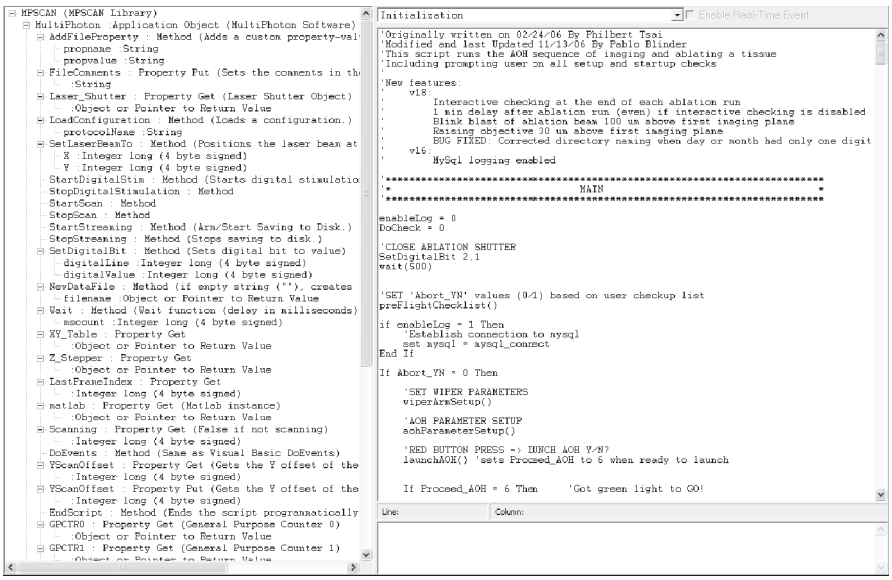


FIGURE 4.2 MPScan user interface and integrated scripting environment. (A) The main window for the display of data; three channels are shown here. Saturation levels are indicated by assigning red to the maximum A/D level. (B) Left panel displays, by default, the Type Library of MPScan; which describes objects and methods available in MPscope to other programs. This allows one to find the objects created in MPScan and to identify their properties and associated remarks. Any other type library can be displayed as well. (C) Right panel display shows the main coding window for control and scripting. The editor shows either the initialization code of the script (equivalent to the "main" function in (D) or event handlers. Events include run-time imaging events and clicks on user-customizable buttons. (D) The Debugging/Messages window. This window is used by the script engine to provide scripting status and debugging notifications. Scripts can also write messages to this window during their execution. (E) Expanded view of the left and right display in panels B and C.

bitmaps. For instance, MPScan uses the Video for Windows Application Programming Interfaces to display frames from an off-screen composing bitmap with a function designed for hardware accelerated streaming video. In principle, the DirectX library, which is geared toward writing video games and is used in e-Maging,[5] should provide even better graphical speed thanks to the ability to perform hardware flips of video surfaces from the main memory to the video card. Although the performance gain is clearly warranted in the case of video-rate two-photon imaging, i.e., order 30 frames per second, it is not clear if the speed gain is worthwhile when imaging is done at our typical rate of 3 frames per second, given the complexity of programming a windowed DirectX application.

As a general design issue, choosing an object-oriented language has clear implications for the architecture of a two-photon microscope program. The cardinal concepts of object orientation, namely, encapsulation, inheritance, and polymorphism, were systematically used in MPScan. This ensures that all end-users would operate the microscope in a consistent manner regardless of the details of the auxiliary hardware that is incorporated into the microscope system, i.e., X-Y translation table, Z-focus stepper, Z-focus piezoelectric objective driver, laser shutter, micromanipulators, etc. To deal with the multiplicity of these devices, MPScan conceptualizes the auxiliary instruments as software objects that are grouped into categories defining their role and common behavior. All device objects are instantiated from a base class called TMPScanDevice; T stands for type, a naming convention to differentiate classes from instances of objects. As shown in Figure 4.3, in the case of X-Y translation table all actual devices descend from the TXYTable class, which implements a common behavior for all translation stages as a set of virtual methods. For instance, when the user clicks on a screen control that moves the objective by a given distance in the X direction, the XYTable object changes the color of a virtual light-emitting diode to red on the front panel, indicating an impending motion, sends the command to perform a relative move, waits for the move to be completed, toggles the light-emitting diode color to green, reads the actual position of the stage, and updates the X position value on screen. To implement a new translation stage, one only needs to create a descendent of TXYTable that will automatically inherit the behavior of the base class. The new class can either add its own methods or override preexisting ones. Encapsulation of properties and methods in each derived classes prevents the unwanted interaction of the code that controls device objects.

Instantiation of objects representing actual devices is done when MPScan starts. MPScan reads the hardware configuration of the microscope, which is stored in the Windows registry. Depending on the settings of the configuration, MPScan creates the device objects from their respective class, e.g., a TXYDMC4040 for a DMC-4040 controller, and attempts to connect them to the hardware they support via their preferred communication interface, e.g., RS-232, USB, TCP-IP, etc., by dynamically loading their manufacturer-provided library if needed. Following their creation and connection, these objects are treated by the rest of the program as instances of their ancestor class using polymorphism; for the case of the DMC-4040 object, the ancestor class is the TXYTable.

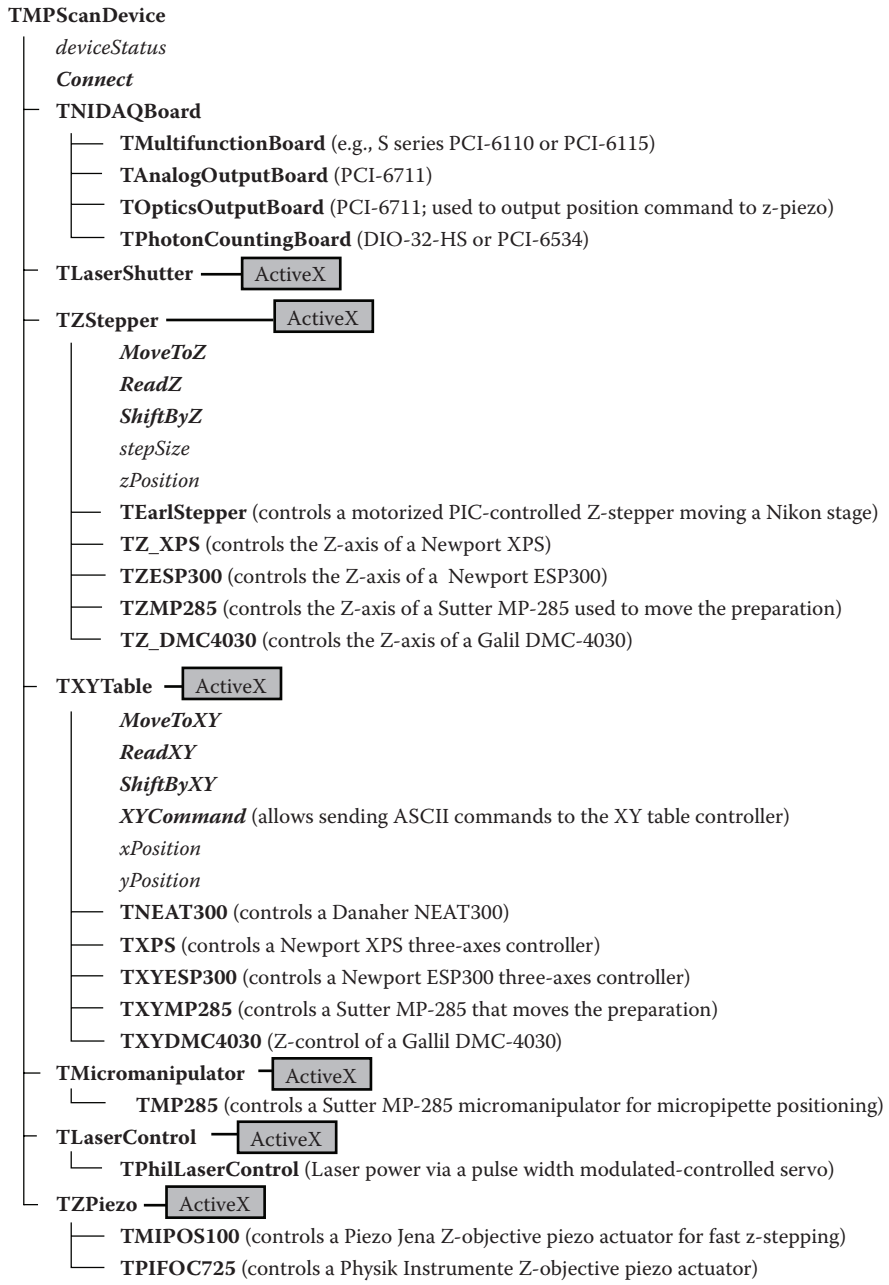


FIGURE 4.3 Class hierarchy of MPScan physical devices. Bold: base classes and descending classes of objects controlling physical devices. Single-inheritance model. *Italic*: properties. *Bold Italic*: virtual methods. For the sake of clarity, many properties and methods have been omitted from this diagram. ActiveX: ActiveX Automation object created by the class for scripting purposes.

4.1.4.4 Componentization

The cornerstone of MPScope interoperability with other programs is the widespread use of Component Object Model (COM) technologies; the reader is referred to References 16 and 17 for background on COM. Although the Microsoft .Net application framework is replacing parts of COM on Windows machines, the need to deploy the .Net 2.0 run-time libraries with any .Net program, and the fact that .Net applications rely on interpretation of bytecodes instead of compiled native code, ruled out the use of .Net as the component foundation of the entire MPScope suite.

The object-oriented design of MPScan provides a coherent architecture to control MPScan programmatically using the COM-based ActiveX Automation standard. Viewed from an outside client, MPScan does not appear as a monolithic program but as an application server consisting of a hierarchy of COM components, some of which represent common operations performed on the microscope and others that abstract a real device connected to the microscope. These components expose an ActiveX Automation interface that can be invoked by COM clients either at run-time, via a late-binding IDispatch interface, or more effectively via direct early-binding v-table function calls that are discovered at compile-time by consulting the Type Library; this library describes objects and methods available in MPscope to other programs and is embedded in the MPScan executable file. An added advantage of using well-proven COM technologies is the possibility to control MPScan over a network from a remote computer, or to output frame data in real-time from MPScan to a remote client through the use of Distributed COM (see Reference 10) providing that there is no firewall between the computers. Furthermore, MPScan can source specific COM events and thus allows a scripting client to be called automatically when scanning starts, scanning ends, or when a frame has been acquired.

The COM-based ActiveX Automation standard is also used extensively in MPScope data files. These files have a fairly unique layout, thanks to the adoption of the Structured Storage technology available on the Windows platform. The organization of MPScope files does not rely on a flat structure but is akin to a directory storing subdirectories and data streams, with each stream containing demultiplexed imaging or analog channel. Gap-free electrophysiology data simultaneously acquired with imaging, and subsequently down-sampled, can be efficiently stored alongside and independently of imaging frames. The use of Structured Storage also considerably eases the writing of analysis programs since they do not have to deal with version-dependent file offsets and pointer calculations to reach a particular piece of data. In addition, imaging metadata are stored in a stream that acts as an extensible header that can be queried and browsed from the Windows Explorer Shell without having to open the file (Figure 4.4). Custom tags can be added to MPScope files in MPScan, e.g., by scripting, without interfering with the basic structure of the file. Overall, using Structured Storage in MPScope facilitates data management as it allows consolidation of all experimental data of a given run into a single file.

To ease interoperability with other programs, MPScope files can be exported as Windows bitmaps, TIFF, or AVI files. In addition, MPView can automatically transfer frames to Matlab™. Another way of opening MPScope files in programs other than MPView is to use the MPFILE.OCX ActiveX component. This add-on, part of

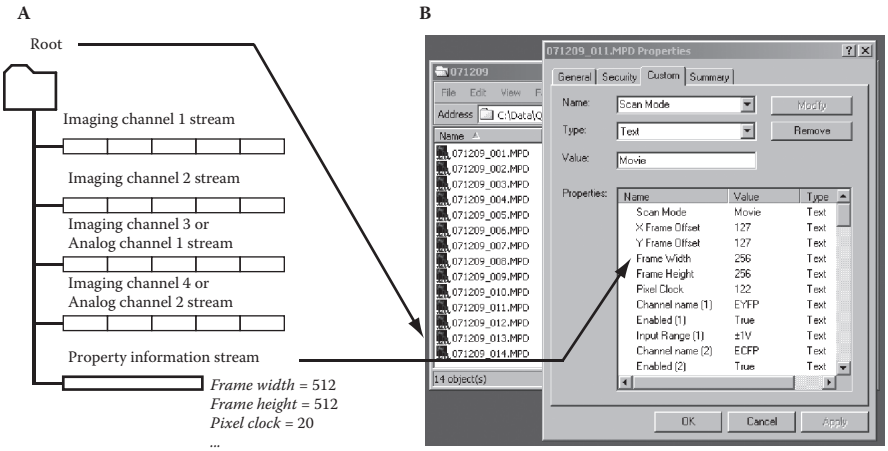


FIGURE 4.4 MPScope file format. (A) Internal layout of a MPScope file. (B) Screen shot of the File Property window, displayed by left-clicking the file name in the Windows graphical shell.

the MPScope suite, provides a simple way of opening MPScope files to obtain frame pixel values or metadata in programs written in languages such as C/C++, Matlab™, Visual Basic, or Delphi™. Until recently, however, MPScope files could only be opened by programs running under Windows. Fortunately, Microsoft has just made the internal format of Structured Storage public,[18] which will allow the development of platform-independent code to deal with MPScope data files. Moreover, several software libraries that are able to read Structure Storage files are available for other operating systems, such POIFS,[19] which is written in Java. The existence of these libraries opens the possibility for analyzing native MPScope data under MacOS X or Linux.

4.1.4.5 ActiveX Scripting

A general problem that is often encountered in microscopy, and a defining feature of optically assisted, plasma-mediated ablation experiments,[3,9] is the repetition of the imaging/ablation cycles and the timely execution of various operations. In principle, each experiment could be automated by modifying the source code of the two-photon acquisition program. Overall, the entire MPScope package represents over 63,000 lines of code. Rebuilding of source code for each application would be inordinately time-consuming and inefficient. The best solution is to avoid source code modification and enable end-users to control the two-photon imaging software with small, external programs written by each user. This approach, termed "scripting," is extensively used in large and complex microscope applications. The prime example of embedded scripting is the inclusion in the Zeiss™ two-photon software of a full-featured programming environment based on the Microsoft Visual Basic for Applications language.

Application scripting requires two software features. The first is a programming language that allows programming scripts to be easily written by end-users. Scripting languages can be mastered in a few hours by nonprogrammers, thanks to a

simple grammar that eschews strongly typed constants, explicit memory allocation, and pointers. To allow users to test their code immediately, scripting languages are usually interpreted. Although interpretation of code with untyped variables leads inevitably to slow execution, speed increase can be achieved by tokenization of the source code before interpretation of bytecodes.

The second feature of scripting is a common protocol between the client scripts and the two-photon server software to pass and retrieve variables and execute commands across process boundaries. On a single Windows computer, client-server communications can be implemented by sending simulated keystrokes or by passing private messages between windows; the latter method is used, for instance, by fluorescence imaging software to control the MDS pClamp electrophysiology program. These communication methods are, unfortunately, extremely rudimentary, do not allow two-way dialogues, and are ill-suited for passing large sets of data. However, adoption of the COM ActiveX Automation standard endows MPScope with well-understood programming entry points with the added possibility of performing Remote Procedural Calls between clients and the two-photon software across a local network via Distributed COM. An added advantage of using ActiveX Automation in scripting is the possibility to use any prewritten ActiveX component that can be either developed with a general-purpose Windows programming language, e.g., C/C++, Visual Basic, Delphi™, etc., or provided by a vendor. For instance, scripts can control an ActiveX component developed in house that interfaces with a 1608-FS USB-based data acquisition board (Measurement Computing, Malboro, Massachusetts). This component adds to MPScan up to eight analog input lines with 16-bit resolution that can be read by scripts.

While Visual Basic for Applications has commendable features, it is too expensive to be used as a scripting language in an academic freeware. However, Microsoft's COM-based ActiveX Scripting technology is a suitable alternative to develop, run, and debug scripts in a language independent manner. Scripting interpreters only need to conform to the ActiveX Scripting specification; in principle, an ActiveX Scripting-enabled application can choose from a wide variety of scripting languages at runtime to execute scripts in a given language. This allows developers to write their scripts either in Basic or Java-derivative languages such as Microsoft Visual Basic for Scripting Edition (VBScript) and Java Script (JScript), Python, or Haskell. The advantage of VBScript is that the VBScript interpreter is free and is available on every copy of Windows as the default scripting engine of Microsoft Explorer Web browser.

MPScan tightly integrates VBScript in its scripting development environment to provide a simple yet powerful way to customize MPScan without having to modify the MPScan source code. Although the scripting development environment features simple editing capabilities with basic debugging capabilities, ten user-customizable buttons provide script writers with a limited Graphical User Interface programming capability. Scripts can be highly complex, especially those automating all optical histology.[3] Figure 4.2 shows such an intricate script that repeatedly performs 256 by 256 μm^2 fast stacks down to 700 μm in depth at greater than 50 different tiled locations in a fixed brain specimen using automatic laser intensity control to ensure consistent brightness of sections located at different depths. The script subsequently controls plasma-mediated ablation of the previously imaged region by opening the

shutter of the ablation laser line and moving the stage in a raster. Following ablation, the script raises the objective, cleans it using a wipe mounted on a servo, and dips it back into the imaging medium for another pass of imaging and ablation. The script runs overnight and generates 45 Gbyte of imaging data. MPScan scripts can, in addition to controlling MPScan, send commands to other applications. For instance, the scripts can communicate with the MySQL database to store or retrieve sets of record via ActiveX Automation.

4.2 MPSCOPE 2.0

4.2.1 General Features

MPScope 2.0 is based on the same design choices of the earlier version, which emphasized fast scanning using symmetrical line scans, intuitive turnkey operation, independence from run-time environment, and a highly scriptable architecture. The analysis program MPView provides common frame operations such as averaging, maximal projection on three axes, several options to export frames, and the possibility to automatically find regions of interest and plot their intensity across time. Most of the new features in MPScope 2.0 are in the acquisition program MPScan. These include:

1. The maximum frame size is extended to 4096 by 4096 pixels from 512 by 512 pixels in the original code, which was adequate for physiology but not for the larger brain areas processed with all optical histology using lenses that have a 4 mm field of view.
2. The recommended option to control the objective and preparation positions uses a model DMC-4040 industrial motion controller from Galil Corporation, which replaces a now obsolete Danaher NEAT300 controller and a custom-built motorized Z-stepper.
3. Support for analog integration and fully digital photon counting on up to four imaging channels to increase the signal/noise ratio under conditions of low emission.
4. Fine control of excitation power through a power attenuation kit from Newport Corporation, which can monitor true root-mean-square laser power via a USB-controlled model 1918-C power meter and implement computer-controlled power feedback by moving a half-wave plate with a model PR50 compact rotation stage via a model SMC-100 controller communicating with the computer using a RS-232 link.[20]
5. Synchronous triggering of the laser pulses within the scan cycle for plasma-mediated ablation.
6. Enhanced support for Z-piezoelectric objective actuators. These devices allow faster Z-motion of the objective position compared with conventional motorized Z-focus controllers but have relatively limited range; thus, a Z-piezoelectric actuator is used together with a Z-focus stepper. At present, MPScope 2.0 supports piezoelectric actuators having a range of either 100 or 400 μm. These include the PiezoJena model MIPOS 100 and Physik Instrumente PIFOC P-725.4C series, which are configured in close-loop

mode to ensure accuracy and repeatability of the motions at the expense of speed. In our laboratory, these piezoelectric actuators are used to generate 3-dimensional X-Z-T movies of 256 by 256 pixels at a rate of 2 frames per second, with 100 μm spans of Z-travel, or 4-dimensional X-Y-Z-T in vivo movies of mouse cerebral vasculature. In the Helmchen Laboratory, these are used to make 4D line-scans that spiral through a depth of cells.[21] Control of these devices from MPScan is straightforward by using a PCI-6711 digital/analog board that provides an analog control voltage proportional to the desired displacement. MPScan can be configured to move the objective either after every line scan or after every frame and to repeat this pattern indefinitely. This allows acquiring tilted frames in addition to the X-Z-T and X-Y-Z-T modes mentioned above. Repeated Z-scanning is achieved by outputting a sawtooth-like command waveform.

4.2.2 Hardware Requirements

The minimal configuration to run MPScope 2.0 consists of:

1. A computer running Microsoft Windows 2000 or XP loaded with the NI-DAQ™ library, the latest version of which can be downloaded from the National Instruments Web site, and at least 1 Gbyte of random access memory. At the time of writing, MPScope 2.0 has not been tested under Microsoft Vista.
2. One National Instruments PCI-6110 data acquisition board.
3. A breakout box with BNC connectors in the front, e.g., National Instruments NI BNC-2090, is highly recommended to interface with the 68-pin connector of the PCI-6110.

For additional hardware control, up to two PCI-6711 analog output boards can be used to send positional commands to a Z-piezoelectric objective actuator and to generate voltage commands for electrophysiological instrumentation. All auxiliary boards have to be connected to the main PCI-6110 board using a National Instrument Real-Time System Integration bus.

4.2.3 Motion Control Support

We use an industrial, off-the-shelf three-axes motion solution based on stepper motors controlled by a commercial 4-channel controller (DMC-4040, Galil Corp., Rocklin, California). Each stage includes a brake, encoders, and limit switches; the wiring for these is given in Figure 4.5 and Tables 4.1 and 4.2. Minimal displacement along the Z axis is 0.3125 μm for our stages, which is smaller than the axial resolution of the microscope. Motion and position reporting commands issued by MPScan are sent to the DMC-4040 controller via Ethernet. Inside MPScan, the DMC-4030 is controlled by its own thread that communicates to the hardware layer via a Dynamic Link Library provided by Galil. Adding DMC-4040 support to MPScan did not require any major software change, since its software object was developed by inheriting its behavior from a virtual, idealized X-Y translation stage and Z-focus stepper.

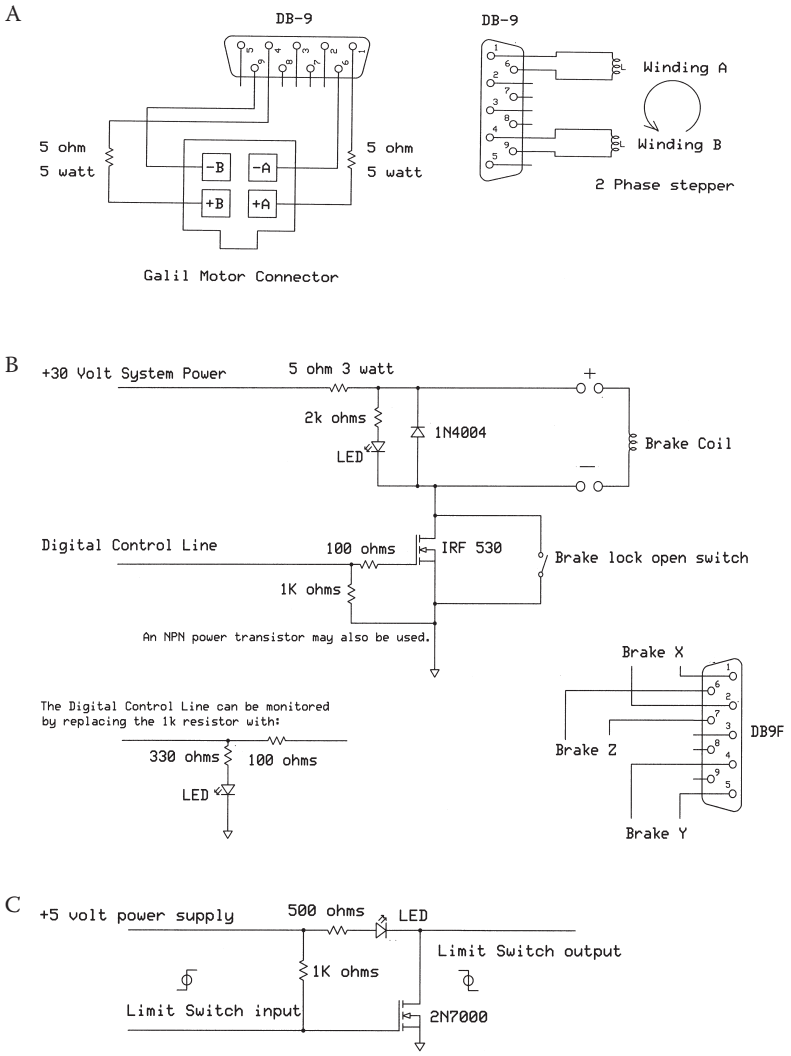


FIGURE 4.5 Schematic of interface circuits for control of X, Y, and Z axes by the Galil DMC-4040. (A) Connections to each of the three stepping motors. The Galil unit provides current pulses to the motor windings, and the voltage can be as high as 30 V. The resistors serve to protect the outputs and also reduce the time-constant of the winding inductances. The DB-connectors serve only as an interface. (B) Connections to each of the three electromagnetic brakes using digital control lines (Table 4.2). The DB-connectors serve only as an interface. (C) Connections to each of the six limit switches (Table 4.2).

As a practical matter, it is useful to maintain manual control of the stages at set-up time. We made use of a three-axis joy stick controller (Feteris S30-JHK-ZT-30R3G-000) with a fourth channel used to read a single-turn potentiometer that

TABLE 4.1
Pin-outs for a Galil DMC-4040 quadrature encoder I/O (15 connector)

Pin		Pin		Pin	
1		6		11	Aux A+
2	B+	7	B–	12	Aux B–
3	A+	8	A–	13	
4	Aux B+	9		14	
5	Ground	10		15	+15 V

Note: Pin-out is for each of up to four separate channels and connectors. Main quadrature encoder is A-B and auxiliary encoder is Aux A–Aux B.

TABLE 4.2
Pin-outs for Galil DMC-4040 external driver digital I/O (44-pin connector)

Pin		Pin	
1		23	Reverse limit switch (Y)
2		24	Reverse limit switch (Z)
3		25	
4		26	
5		27	DO-2 (brake Y)
6	LS-Comm (+5 V)	28	
7		29	
8		30	+5 V
9		31	Ground
10		32	
11	O-PWR (+5 V)	33	
12	DO-3 (brake Z)	34	
13		35	Ground
14	O-return (ground)	36	Forward limit switch (X)
15	+5 V	37	Forward limit switch (Y)
16		38	Forward limit switch (Z)
17		39	
18		40	Ground
19		41	DO-1 (brake X)
20		42	
21		43	
22	Reverse limit switch (X)	44	

Note: Pin-out is for all four channels on a single connector.

sets the scale for the speed of movement. The DMC-4040 contains an 8-channel analog to digital converter port that is convenient for this task; the wiring is shown in Figure 4.6. Firmware code for joystick resides in the DMC-4040; see Appendix. The manual controller is overridden by MPScope to prevent conflicts while the program is running.

4.2.4 Analog Integration

Photomutiplier Tube (PMT) currents are converted into voltage signals by preamplifiers, which can be either purchased commercially or, in our case, built in the laboratory. These preamplifiers are either of the transimpedance type or consist of a voltage preamplifier measuring the voltage drop caused by the PMT current output across a grounded resistor. Numerous companies produce preamplifiers suitable for PMT operations, i.e., Stanford Research Systems in the United States and Femto in Germany.

Simple signal conditioning of the preamplifier output can performed either by digital integration or by low-pass filtering. The first method consists of sampling the preamplifier analog signal several times during the pixel duration and adding these values. Drawbacks of this method include an increase in samples collected and more importantly the inability to deal with fast preamplifiers. For instance, the transimpedance preamplifiers in our laboratory output a voltage pulse lasting 50 to 100 ns in response to a single photon. These responses are too fast to be adequately digitized by the analog/digital converters of the PCI-6110, which have a maximal sampling frequency of 5 MHz. The alternative of low-pass filtering the PMT signal can be done either in the preamplifier module if this option is built in, or by an external low-pass filter unit. In either case, the cutoff-frequency f_c of the filter should be set to $f_c = 2/(\pi\tau_p)$ where τ_p is the pixel integration time.[22]

An efficient analog signal conditioning solution is to use a true pixel integrator. This device offers several advantages over low-pass filtering:

1. Suppression of correlation between adjacent pixels
2. Simplicity of operation, since there is no need for cut-off frequency adjustment whenever the pixel dwell time changes
3. Better image quality thanks to an overall variance of the image as much as twice as small as that obtained with low-pass filtering[20]

A further refinement of true pixel integration consists of using alternatively switched integrators to prevent blanking-off time when one integrator needs to be reset. This scheme, known as multiphase pixel integration, is used in commercial confocal microscopes such as the Nikon C1 with the name of Dual Integration Signal Processing.

A challenge in designing a stand-alone multiphase pixel integrator is to generate the timing signals that switch alternate integrators and reset them in synchrony with the pixel clock. Fortunately, the PCI-6110 board, under MPScan control, can output several signals that can be combined to control a multiphase integrator. The schematics of a simple multiphase pixel integrator in use in our laboratory can be found in Figure 4.7. In brief, each channel of this circuit consists of a Linear Technologies

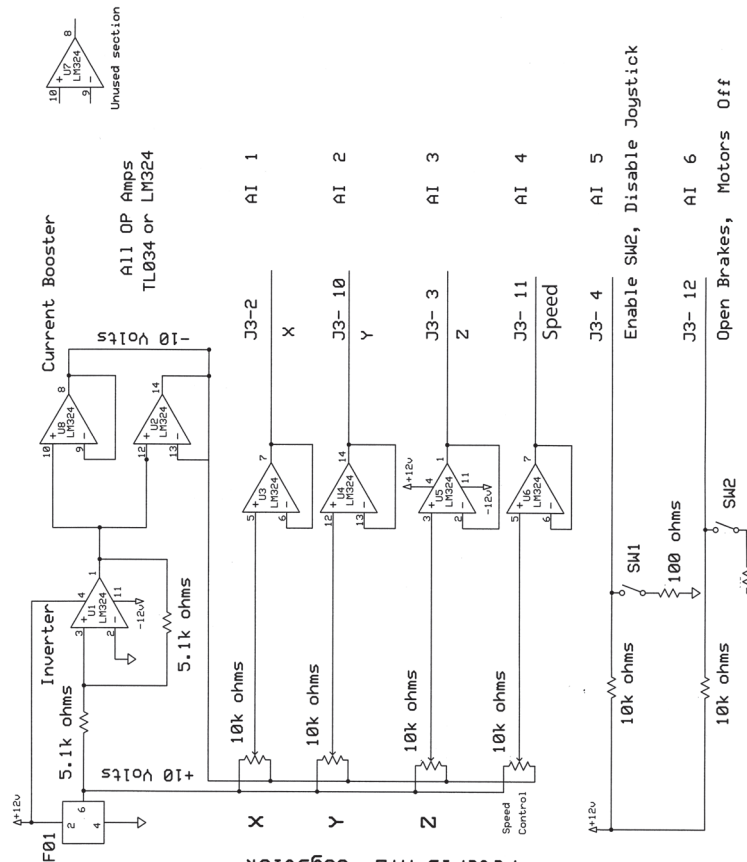


FIGURE 4.6 Schematic of the circuit for a resistive 3-channel joystick to control X, Y, and Z axes. The joystick is energized by power from the Galil DMC-4040 and the output of each potentiometer is read by analog-to-digital converters in the Galil unit and used to control the position of the stages; see Appendix for code. A fourth potentiometer is read to set an overall speed level and a fifth and sixth are utilized as binary switches since there are no digital inputs on the analog input connector to the Galil unit.

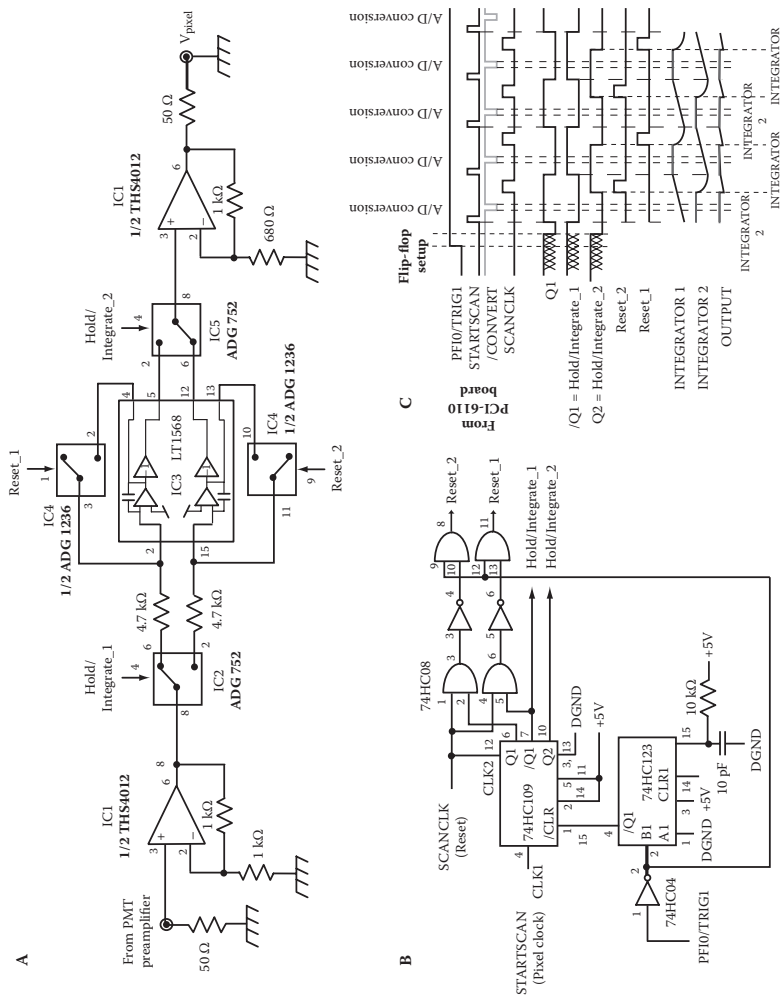


FIGURE 4.7 Multiphase pixel integrator schematics. (A) Multiphase pixel integrator for one imaging channel. (B) Control logic. (C) Timing diagram of the control logic. The /CONVERT signal is not used by the control logic but is added to show that the actual digitization window occurs between the STARTSCAN and SCANCLK signals.

LTC1568 filter chip. This integrated circuit features two operational amplifiers with bandwidth of 50 MHz each coupled to its own capacitor. The two capacitors, with a nominal value of C_f = 105 pF, are conveniently matched to within 0.5 %. In addition, each op amp can be followed by an inverter contained in the same package. A dual voltage integrator with an output of $V_{out} = (R_f C_f)^{-1} \int_{T_p} V_{in}\, dt$, where T_p is the pixel duration and V_{in} is the input voltage, is created by adding a resistor R_f = 4.7 kΩ before each of the LTC1568 inputs. Each integrator is reset by shortcircuiting its capacitor using an Analog Device ADG1236 dual switch, which has a low charge injection. Switching between integration and reset is performed by two ADG752 ultra-fast switches with t_{on} = 8 ns and t_{off} = 3 ns. Input and output signal conditioning is performed by a Texas Instrument THS4012 dual op-amp. TTL circuits that control the switches receive their inputs from the PFI0/TRIG1, STARTSCAN, and SCANCLK signals issued by the PCI-6110 board (Figure 4.7A). PFI0/TRIG1 is used to reset the 74HC109 flip-flops at the beginning of an imaging session via a 74HC123 one shot, while STARTSCAN and SCANCLK create the Integrate/Hold and Reset pulses (Figure 4.7B). The control logic can issue the timing signals simultaneously to several banks of integrators, each assigned to an imaging channel.

A refinement of integration that is appropriate for low light levels is the hybrid photon counting technique.[2,22,23] In this method, the input of the integrator consists of short TTL pulses triggered by single photon pulses from the PMT outputs, which are discriminated by amplitude thresholding to both remove PMT dark noise and regularize the amplitude of the photoelectric pulse.

4.2.5 Dissemination and Support

As was previously the case with MPScope 1.0, MPScope 2.0 is available free of charge to academic users by contacting the authors. MPScope users have also the possibility to become members of the MPScope Google discussion group, which deals with scripting, recompilation, optoelectronics, and hardware support issues. In addition, MPScope 2.0 can be found at the NeuroImaging Tools and Resources Clearinghouse (NITRC) Web site (http://www.nitrc.org/) administered by the National Institutes of Health. Continuous development of MPScope by the authors and the MPScope community will ensure that MPScope remains compatible with the latest technological changes, including the possible replacement of galvanometric mirrors with faster scanning devices, such as acousto-optic modulators, and the incorporation of galvanometric mirrors or acousto-optic modulators for additional laser beams to be used for ablation or uncaging.

ACKNOWLEDGMENTS

We thank T. Baldacchini of the Technology and Applications Center, Newport Corp., Irvine, California, for the loan of a variable laser attenuator kit and P. Blinder and P. S. Tsai for the example script of Figure 4.2. This work was supported by the NIMH (MH071566), NIBIB (EB003832), NCRR (RR021907), and NSF (DBI 0455027).

APPENDIX

Firmware for Galil DMC-4040 controller to operate three-axis stage through an analog joystick

```
#AUTO                          ;'allows the program to execute
                                 when power is applied
#A                             ;'this section initializes the
                                 stepper motor parameters
MT -2.0,-2.0,-2.0
YA 16,16,16                    ;'16X stepper resolution
YB 400,400,400                 ;'basic pulses per revolution
SP 3000,3000,3000              ;'speed in rpm
LD 0,0,0                       ;'enable limit switches
CN -1,-1,-1,0,0                ;'limit switch active low
DP 0,0,0                       ;'set initial stage position to
                                 zero
VX=0                           ;'zero all variables
VY=0
VZ=0
VXA=0
VYA=0
VZA=0
OB 1,0                         ;'brakes on X axis
OB 2,0                         ;'brakes on Y axis
OB 3,0                         ;'brakes on Z axis
#B                             ;'start of main loop
  VMIN = .5                    ;'minimum acceptable value from any
                                 joystick axis
  V4 = @AN[4]                  ;'read the "Rate", i.e., multiplier
                                 potentiometer value
  VMLT = V4*V4*5               ;'value from the pot is adjusted
                                 for the X and Y axes
  VMZ = V4*V4                  ;'value is adjusted for the Z axis
  VS1 = @AN[5]                 ;'read limit switch 1
  VS2 = @AN[6]                 ;'read limit switch 2
  JP #C, (VS2>VMIN)            ;'if VS2 high, continue normally,
                                 otherwise check VS1
  IF (VS1<VMIN)
    OB 1,1                     ;'if VS1 low, then release all
                                 brakes
    OB 2,1
    OB 3,1
    MO                         ;'turn off motors
  ELSE                         ;'if VS1 high, apply all brakes and
                                 enable motors
    OB 1,0
    OB 2,0
```

```
  OB 3,0
  SH                          ;'turn on motors (servo here)
ENDIF
JP#B                          ;'continue checking switches until
                               VS1 is high
#C                            ;'if reached, apply all axis brakes
OB 1,0
OB 2,0
OB 3,0
SH                            ;'turn on motors just in case
#XAXIS
  V1=@AN[1]                   ;'read X joystick value
  V1A = @ABS[V1]              ;'calculate absolute value
JP #XEND, V1A<VMIN            ;'if below VIN then clear value,
                               set brake, stop motion
JP #XAXIS1,(_LFX=0)&(V1<0)    ;'OK to move in X if positive
                               switch set and X negative
JP #XAXIS1,(_LRX=0)&(V1>0)    ;'OK to move in X the negative
                               switch set and X positive
JP #XEND,(_LFX=0)|(_LRX=0)    ;'stop X motion for any other case
#XAXIS1
  VX=V1*VMLT                  ;'if X more than VMIN then multiply
                               value by VMLT
  OB 1,1                      ;'release X brake
  JG VX,0,0                   ;'set Jog mode for the X axis with
                               the VX value
  BGX                         ;'begin X motion
JP #XAXIS                     ;'cycle until joystick is released
#XEND                         ;'end of X axis operation
  VX = 0                      ;'clear X value
  OB 1,0                      ;'apply X brake
STX                           ;'stop X motion
#YAXIS                        ;'begin to process Y and Z axes
                               same as X axis
  V2 = @AN[2]
  V2A = @ABS[V2]
JP #YEND, V2A<VMIN
JP #YAXIS1,(_LFY=0)&(V2<0)
JP #YAXIS1,(_LRY=0)&(V2>0)
JP #YEND,(_LFY=0)|(_LRY=0)
#YAXIS1
  VY = V2*VMLT
  OB 2,1
  JG 0,VY,0
  BGY
JP #YAXIS
#YEND
  VY = 0
```

```
    OB 2,0
    STY
  #ZAXIS
    V3 = @AN[3]
    V3A = @ABS[V3]
  JP #ZEND,V3A<VMIN
  JP #ZAXIS1,(_LFZ=0)&(V3<0)
  JP #ZAXIS1,(_LRZ=0)&(V3>0)
  JP #ZEND,(_LFZ=0)|(_LRZ=0)
  #ZAXIS1
    VZ = V3*VMZ
    OB 3,1
    JG 0,0,VZ
    BGZ
  JP #ZAXIS
  #ZEND
  VZ = 0
  OB 3,0
  STZ
JP #B                          ;'loop through X, Y, and Z axes
#LIMSWI                        ;'limit switch interrupt processing;
                                 all 6 switches are checked
IF (_LFX=0)                    ;'check if forward X switch is
                                 asserted
   #TSTFX
   JS #WAITX                   ;'wait for joystick X channel to be
                                 released
ENDIF
IF (_LRX=0)                    ;'check if reverse X switch is
                                 asserted
   #TSTRX
   JS #WAITX
ENDIF
IF (_LFY=0)                    ;'check if forward Y switch is
                                 asserted
   #TSTFY
   JS #WAITY
ENDIF
IF (_LRY=0)                    ;'check if reverse Y switch is
                                 asserted
   #TSTRY
   JS #WAITY
ENDIF
IF (_LFZ=0)                    ;'check if forward Z switch is
                                 asserted
   #TSTFZ
   JS #WAITZ
ENDIF
```

```
IF (_LRZ=0)                    ;'check if reverse Z switch is
                                 asserted
#TSTRZ
   JS #WAITZ
ENDIF
ZS 1                           ;'reset stack pointer, cancel
                                 interrupt by reducing stack by
                                 one
JP #B                          ;'return to main loop
#WAITX                         ;'subroutine to watch X chan.
                                 joystick; returns when X below
                                 VMIN
   VX1 = @AN[1]
   VXAB= @ABS[VX1]
JP #WAITX,(VXAB > VMIN)
EN
#WAITY                         ;'subroutine to watch Y channel
                                 joystick
   VY1 = @AN[2]
   VYAB = @ABS[VY1]
JP #WAITY, (VYAB > VMIN)
EN
#WAITZ                         ;'subroutine to watch Z channel
                                 joystick
   VZ1 = @AN[3]
   VZAB = @ABS[VZ1]
JP #WAITZ,(VZAB > VMIN)
EN
```

REFERENCES

1. Nguyen, Q.-T. et al., MPScope: A versatile software suite for multiphoton microscopy, *Journal of Neuroscience Methods*, 156, 351–359, 2006.
2. Tsai, P.S. and Kleinfeld, D. In vivo two-photon laser scanning microscopy with concurrent plasma-mediated ablation: Principles and hardware realization, in *Methods for* In vivo *Optical Imaging,* 2nd edition (Frostig, R. D., Ed.), CRC Press, Boca Raton, in press, 2008.
3. Tsai, P.S. et al., All-optical histology using ultrashort laser pulses, *Neuron*, 39, 27–41, 2003.
4. Nikolenko, V. et al., A two-photon and second harmonic microscope, *Methods*, 30, 3–15, 2003.
5. Nguyen, Q.-T. et al., Construction of a 2-photon microscope for real-time Ca^{2+} imaging, *Cell Calcium*, 30, 383–393, 2001.
6. Rietdord, J. and Stelzer, E.H.K., Special optical elements, in *Handbook of Biological Confocal Microscopy,* 3rd ed. (James B. Pawley, Ed.), Springer, New York, 2006, pp. 43–58.
7. Pologruto, T.A. et al., ScanImage: Flexible software for operating laser scanning microscopes, *Biomedical Engineering Online,* http://www.biomedical-engineering-online.com/content/2/1/13, 2003.

8. Tsai, P.S. et al., Principles, design, and construction of a two-photon laser-scanning microscope for in vitro and in vivo brain imaging, in In vivo *Optical Imaging of Brain Function* (Frostig, R. D., Ed.), CRC Press, Boca Raton, 2002, pp. 113–171.
9. Nishimura, N. et al., Targeted insult to individual subsurface cortical blood vessels using ultrashort laser pulses: Three models of stroke, *Nature Methods,* 3, 99–108, 2006.
10. Nguyen, Q.-T. and Miledi, R., e-Phys: A suite of electrophysiology programs integrating COM (Component Object Model) technologies, *Journal of Neuroscience Methods*, 128, 21–31, 2003.
11. Dempster, J., *The Laboratory Computer: A Practical Guide for Physiologists and Neuroscientists* (Biological Techniques Series), Academic Press, San Diego, 2001.
12. Oney, W., *Programming the Microsoft Windows Driver Model*, Microsoft Press, Redmond, 1999.
13. PCI E Series Register-Level Programmer Manual, National Instruments, 1998.
14. NI-DAQ User Manual for PC Compatibles, National Instruments, 2000.
15. DAQ-STC Technical Reference Manual, National Instruments, 1999.
16. Brockschmidt, K., *Inside OLE*, Microsoft Press, Redmond, 1995.
17. Box, D., *Essential COM* (The DevelopMentor Series), Addison-Wesley, Reading, 1998.
18. Windows Compound Binary File Format Specification, Microsoft Open Specification Promise, Microsoft Corporation, 2007.
19. Apache POI-POIFS-Java implementation of the OLE 2 Compound Document format. The Apache POI Project. http://poi.apache.org/poifs. 2007.
20. Computer Controlled Variable Attenuator for Lasers, Application Note 31, Newport Corporation, 2007.
21. Göbel, W. et al., Imaging cellular network dynamics in three dimensions using fast 3D laser scanning, *Nature Methods,* 4, 73–76, 2007.
22. Art, J., Photon detectors for confocal microscopy, in *Handbook of Biological Confocal Microscopy,* 3rd ed., (James B. Pawley, Ed.), Springer, New York, 2006, pp. 251–264.
23. Driscoll, J. et al., Photon counting and gating for improved detection in two-photon microscopy (in preparation).

26 February 2009

FINAL CORRECTIONS TO: "MPScope 2.0: *A Computer System for Two-Photon Laser Scanning Microscopy with Concurrent Plasma-Mediated Ablation and Electrophysiology"*

Author list on content page "v"
From *Quoc-Thang Nguyen, Earl M. Dolnick, Jonathan Driscoll, and David Kleinfeld*
To *Quoc-Thang Nguyen, Jonathan Driscoll, Earl M. Dolnick, and David Kleinfeld*

Page 123, paragraph 2, line 13; Section 4.1.4.2
From "… least 8 Gbytes …"
To "… least 2 Gbytes …"

Page 124, caption to figure 4.2
From "… (equivalent to the "main" function in (D) …"
To "… (equivalent to the "main" function in a program written in the language C) …"

Page 129, paragraph 3, line 10; Section 4.1.4.5
From "… available on …"
To "… available in …"

Page 134, paragraph 3, line 1; Section 4.2.4
From "… can performed …"
To "… can be performed …"

Page 137, paragraph 2, line 2; Section 4.2.4
From "… technique.[2,22,23] "
To "… technique.[22,23] "

Page 142, reference 23
From "Photon …"
To "Gigahertz photon …"