document version 3.0
last updated 14 September, 2005

The CORTEX (version 5) User's Manual

Laboratory of Neuropsychology, NIMH

TABLE OF CONTENTS

1. Introduction

1.1 Cortex: A Program for COmputerized Real Time EXperiments.

Cortex is a program for running neurophysiological and behavioral experiments on a PC. Although it is still undergoing revisions, it is a working program that we have been using to collect data for more than eight years. The integrity of the data, timing accuracy, etc. has all been validated. Cortex runs behavioral paradigms designed -+by the user, displays visual stimuli to the subject, collects neurophysiological and behavioral data, monitors and stores eye position data, and shows the results to the user during the experiment. Because Cortex has the capability to display and manipulate any image that has been previously created and saved on disk by the user, the variety of visual stimuli it can present is practically unlimited. The program is oriented toward experiments in vision, but it can be used in a variety of other contexts.

Data acquisition capabilities include (dependent on the available I/O hardware) up to 40 spike inputs (TTL +5v digital inputs), specialized functions for monitoring hand-bar and lever movements (TTL +5v digital inputs), eye-movement monitoring functions (A/D inputs), keyboard, mouse, and touchscreen inputs. Reward triggering for alert-subject experiments is accomplished through TTL +5v digital output. The user retains access to the full-array of possible uses for the I/O equipment on-board (such as D/A output, or specialized digital I/O) with easy-to-use C functions. Up to five data acquisition boards, one of each type, may be accessed by CORTEX.

Graphical capabilities include a device-generalized graphics kernel that allows the user to run the same experiments on any of the supported configurations. To run in the single-computer configuration, either a Number Nine Co. Sgt. Pepper board, or a #9GXi-TC graphics board is required. Since these two graphics boards are no longer manufactured, a two-computer version of Cortex was developed. With the two-computer version, any VESA compliant graphics card/monitor, which is supported by Scitech Display Doctor or Microsoft DirectX, may be used. Many of the low-level technical aspects of graphical programming are handled directly by CORTEX; thus the graphical programming environment is relatively simple to use.

CORTEX is designed to simplify combined behavioral and neurophysiological experimentation by directly handling trial randomization and simple staircasing. The user retains complete control of the trial staircasing, if desired, and can generate any staircasing paradigm needed. The user can even change the staircasing design on-line, in any way, based on behavioral responses.

Associated with Cortex are a number of utility programs. Shipped with Cortex, in the \REMOTE32 and \UTIL_BIN directories, is the DOS program, CORTVIEW.EXE. This program can be used to view the contents of the Cortex output data file. Rather than viewing the results, this program also can be used to pipe the output to a text file. From the Cortex web site, there are links to other Cortex data analysis programs. One program that was written at the Laboratory of Neuropsychology (NIMH) is the Cortex Windows Workbench. This program provides an interactive and graphical look at Cortex data files. Cortex files can also be analyzed using PCOFF, which was developed in the Laboratory of Neurophysiology (NIMH). PCOFF is a comprehensive event code sequence search program that supports graphical display of rasters, histograms, reciprocal interval plots, using a number of different schemes to align data on specific events. The program also does analog signal processing (e.g. integration, averaging, etc.) as well as non-parametric and descriptive statistics. The Laboratory of Neurophysiology recently contributed a program called MatOFF. It is a powerful event search and data plotting program based on their earlier PCOFF program. Matoff runs under Microsoft Windows using MATLAB. Other laboratories at NIH, the University of Verona, MIT, and the Salk Institute have also contributed Matlab routines to help with the analysis and use of Cortex. These Matlab routines can be accessed from links on the Cortex web page.

1.2 Getting the latest version of Cortex

The latest code, executables, and documentation can be downloaded from the Cortex web site:

www.cortex.salk.edu

1.3 Technical Support

If you are experiencing problems with Cortex which are not covered in the Cortex documentation, please post a message to the Cortex web site's discussion group. The discussion group can be accessed from the "Discussion" page of the Cortex home page (http://www.cortex.salk.edu/phpBB2).

1.4 History (you may skip this part)

CORTEX has been gone through so many transitions over the decades that it is daunting to list everyone who has contributed to its development. Dr. Robert Desimone, who has overseen and guided the development of CORTEX since he initially ported it from the PDP-11 platform over a decade ago, made a valiant effort to do so several years ago. This is quoted below.

- **Thomas M. White** - the lead programmer for CORTEX from 1989-1997. Re-wrote CORTEX from scratch, implemented the in-line compiler (the CORTEX state system - CSS), real-time multi-threaded data collection system, and support for multiple touch screens, graphics boards, and data acquisition systems. He also wrote the initial procortex, Stats100, and Grast utility programs. Upon leaving his work on CORTEX he went on to earn his MD degree from Cornell University.
- **Trina M. Norden-Krichmar** - took over the role of lead programmer from Tom White in the fall of 1997. She added a Windows DirectX receive program, support for DirectX sound and movies, and support for PCI data acquisition boards. She is currently working on the implementation of a Windows version of the send program, supporting the DOS version, and updating the documentation.
- **Jessica Benson** – rejoined the Cortex Team in the summer of 2004 after a year absence and is back working on device drivers, Install Shield, updating the website and documentation, fixing bugs and testing. She worked on the Cortex Team from the summer of 2001 to the summer of 2003 writing the Visual C++ and MFC code to transform Cortex into the wonderful dialog boxes and menus that you see today the windows version.
- **Eric Boulden** – worked on the Cortex Team from the summer of 2003 to the summer of 2004. He converted the old-fashioned Cortex web site into a slick PHP/MySQL database driven web site. He also added an InstallShield-based installation to VCortex, converted the drivers for the Random Spike Device and PCI-DIO24 board for Windows 2000 and XP version, and was involved in bug fixes, testing, and documentation.

Others who have contributed substantially to CORTEX include:

- Dr. Steve Macknik - wrote the TIGA and MGL graphics modules, as well as the User's manual.
- Dr. Andrew Mitz - helped write the technical manual, contributed several user-accessible functions to CSS, and contributed the PCOFF and MatOFF data analysis programs.
- Dr. Jamie Mazer - contributed the Set module and several user-accessible functions to CSS.
- Dr. Earl Miller - overhauled the Grast program for graphical analysis of CORTEX data files, but now has contributed Matlab versions of these analysis tools.
- Robert Baumann - developed the CORTEX Windows Suite - a set of tools to facilitate the analysis of CORTEX data files.
- Dr. Giuseppe Bertini - helped design the Cortex web pages, especially the framed function reference, and contributed Matlab data analysis routines.
- David Mechner – wrote the code to support sound in the DOS version of Cortex.

Dr. Robert Desimone wrote the following for the 1997 version of the Cortex User's Manual:

It is currently impossible to list all of the people who have helped in the development of CORTEX, through their suggestions, testing, and bug-reports. The section below acknowledges those who were most helpful in CORTEX's early years.

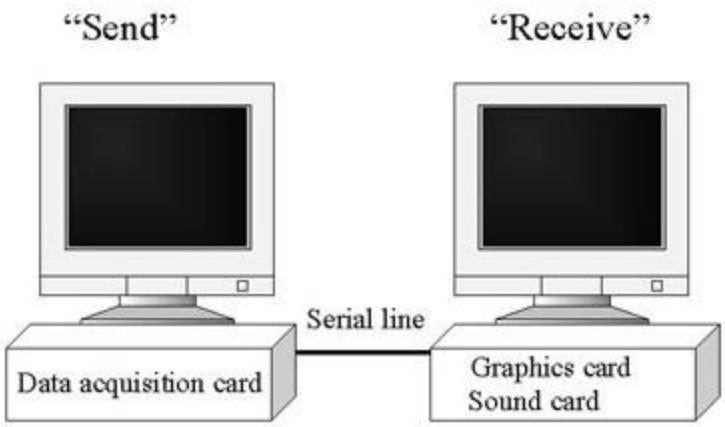
The remote ancestor of Cortex can be traced to an assembly language program for collecting spike data and controlling an optical bench, written for the PDP-12 by David Bender in Charlie Gross' lab at Princeton during the early 1970's. The PDP-12 was a state-of-the art laboratory computer with 8K of core memory (and a 12-bit word), a bit-mapped graphics display, a real-time programmable clock, and analog and digital interfaces built in. After Dave left Princeton for Buffalo, I assembled the necessary components for a PDP-11 system (which, shockingly!, had no lab interfaces built in) and sketched the requirements for a new spike collection program that could make use of the luxurious 64K of memory (and 2.5 MB hard disk!) that was then available. The core of this program was written in 1980 in Whitesmith's C by a moonlighting mathematician, Phil Thrift, and a high school student who had been programming in C since age 12. When I left Princeton for NIMH, the program branched in its development. One branch continued to be developed at Princeton by Tom Albright, and the other branch developed at NIMH. Jeff Moran at NIMH significantly rewrote the program, then called "Behave", and added modules for controlling the subject's behavior and controlling an external Jupiter Graphics system for presenting stimuli.

As to be expected, a memory size that once seemed luxurious eventually became ridiculously too small. After the introduction of the IBM-AT, Stan Schein and I decided to switch from the PDP-11, in part because of the great difference in price and in part because the C compilers for the AT allowed transparent access to 640K of memory, whereas it was difficult to use more than 64K on the PDP-11. A Pepper SGT graphics card from Number Nine Computer was eventually chosen to replace the Jupiter graphics system. Stan, who moved from NIH to Harvard Medical School, hired Thuan Tran, then a computer science student at MIT to port the program after I had naively (and innocently!) assured Stan that the porting would take at most six weeks, once we understood all of the new hardware. Because the old program was inflexible and filled with kludges to get around memory limitations, Thuan did not actually port it. Although he retained a few design aspects of the original PDP-11 program, he completely rewrote and greatly expanded the program over the course of the next couple of years after graduating from MIT, with direction on the design by Stan and myself and support from Stan's grant (R01-EYO6096).

In 1989, Thuan took a full time job with a computer company but continued to work part-time on the state-system interface to Cortex, on contract to NIMH. At about the same time, I hired Tom White at NIMH, and he significantly rewrote most of Cortex again, in order to make it more user friendly, to greatly upgrade its stimulus display capabilities, and to improve the maintainability of its source code. He also took over the development of GRAST, a histogram display program, and Procortex and STATS100, numerical analysis programs, greatly expanding and rewriting them over the course of the past year. Tom integrated STATS100 with SYSTAT, a commercial statistical program, and wrote a data analysis compiler that users can use to write their own analysis routines, without doing any actual programming. Tom is now at Cornell Medical School in a MD/Ph.D. program, but continues to work on Cortex occasionally, out of the goodness (truly) of his heart. I myself work on Cortex code, in my spare milliseconds, and am reasonably familiar with the workings of most of the modules. Rosalyn Merrill (NIMH), Amir Geva, and Jeff Moran also worked on small pieces of code for either Cortex or GRAST at one time during their development. Most recently, Marcello Gattass, the brother of Ricardo Gattass, has taken over the development of GRAST for the time being. Marcello promises a MS Windows version of GRAST sometime in the future. Steve Wise at NIMH has contributed towards some of the considerable development expense of Cortex. Finally, Cortex has benefited from the suggestions (and, unfortunately, bug-finding) of many of its users, particularly Mark Wessinger, Lin Li, Sidney Lehky, John Duncan, Earl Miller, Jennifer Hart, Driss Boussaoud, Josef Rauschecker, Ricardo Gattass, Leonardo Chelazzi, Andy Mitz, Richard Jeo, Carl Olson, Rebecca Hoag, and Peter DeWeerd.

2. Hardware and Software Requirements

2.1 Two-computer Cortex (REMOTE32)



2.1.1 Basic "Send" computer

The Cortex "send" computer controls experimental timing, data collection, and sending commands to the graphics ("receive") computer. The "send" computer contains the data acquisition card.

A Pentium computer (at least 100 MHz) with 32 MB RAM and at least a 1 GB hard drive for data storage is recommended. These specifications are based on the assumption that the computer will have Win95/Win98 installed on it also.

2.1.2 Basic "Receive" computer

The Cortex "receive" computer displays the graphics to the subject. There are two versions of the "receive" program: a DOS Scitech Display Doctor (SDD) based version and a Windows DirectX based version.
For the SDD DOS receive program, the computer must contain a Scitech Display Doctor supported graphics card.

For the DirectX receive program, the computer must contain a DirectX supported graphics card.
Like the send computer, the receive computer should also be a Pentium comp uter with at least 32 MB RAM. Since graphics are more CPU intensive, if you have two unequal computers, the faster computer with more memory should be the receive computer.

2.1.3 Operating Systems

The send computer and DOS Scitech Display Doctor receive computer must be booted up in true DOS to run with accurate timing. A computer that has Win95 or Win98 can be booted up into DOS easily. WinNT and Win2000 can not be easily booted up into DOS, so I do not recommend it for the DOS based Cortex program.

The Windows DirectX receive program must be run from Windows. It can run with Win95, Win98, Win2000 or XP. WinNT definitely can not be used for the DirectX receive program, since it does not support DirectX 5.0 or higher.

2.1.4 Graphics cards

A good graphics card is only required in the receive computer. It must be supported by the Scitech Display Doctor for the DOS SDD receive program, or it must be supported by DirectX for the Windows DirectX version (see section 2.2.10 below)

A graphics card with at least 4 MB of onboard video memory is recommended.

2.1.5 Data acquisition cards

All of the data acquisition boards originally specified for the single computer version of Cortex should also work with the dual version of Cortex. From Keithley/Metrabyte, there are: DASH-16 (for analog and digital I/O), and the PIO24 or PIO96 (for digital I/O only). From ComputerBoards, there are the clones: CIO-DAS16, CIO-AD16, CIO-DAS1602/12, CIO-DIO24 and CIO-DIO96.

The ComputerBoards PCI-DAS1602/12 and PCI-DIO24 boards are also supported, in case your computer does not have ISA slots.

When you are buying your data acquisition board, you may also want to order a screw terminal interface. The Metrabyte "Universal Screw Terminal Accessory Board STA-U" works well with the CIO products.

The PCI-DAS1602/12 requires a 100-pin connector and interface. From the ComputerBoards, Inc. catalog, the parts are listed as a C100FF cable and a CIO-TERM100 screw terminal board. The CIO-DIO24 and PCI-DIO24 require a 37-pin cable and MINI-37 screw terminal board.

For use with all data acquisition boards, the screw terminal interfaces must be altered (flip-flops must be added) or have another device that performs the latching, in order to collect spike data. The Cortex web page contains the necessary diagrams under the heading "Spike Flip-Flop Circuitry and Thalamus".

2.1.6 Sound cards

2.1.6.1 DOS (rvesactx.exe) version

A SoundBlaster card must be installed in the receive computer. Any modern SoundBlaster card should work (e.g., SB16, AWE32, or AWE64). Problems have been reported on SB16 compatible cards.

2.1.6.2 DirectX (dxrecv.exe) version

Any sound card which is supported by DirectX can be installed in the receive computer.

2.1.7 Touch screens

The code was written for and tested on a Microtouch touchscreen. The latest Microtouch Touchware driver software is available from the 3M web site (http://www.3m.com/3MtouchSystems/downloads/legacy.jhtml). You must install the DOS version of the Touchware driver on the send computer.

2.1.8 Serial connection

The two computers are connected via a serial cable between COM ports. The serial cable connecting the two computers must be a null modem cable. (Laplink-style cables will work.) Black Box Corporation sells a "Universal Serial cable (DB25F/DB9F to DB25F/DB9F)", and its part number is BC01802, which works well with Cortex.

2.1.9 Monitors

A good monitor is only required on the receive computer. Remember that the highest refresh rate that can be achieved is dependent on the graphics card and the monitor. Therefore, when selecting a monitor, make sure

that the monitor can handle the refresh rate at the desired resolution and color depth. You will also want to buy a video-splitter (such as a VOPEX splitter from NTI) so that the graphical output can be displayed on a monitor for the subject and a monitor in your setup.

2.1.10 Software

SciTech Display Doctor must be installed on the receive computer only if you are going to use the DOS Scitech Display Doctor receive program. A free copy, can be downloaded from the SciTech web page http://www.scitechsoft.com/products/ent/free_titles.html. Please note that the Scitech Display Doctor is no longer supported by Scitech. Refer to the Scitech web site to find out which graphics cards are supported.

In order to use the DirectX receive program, the DirectX 6.0 (or higher) driver must be loaded. The DirectX driver is freely available from the Microsoft web site. If you are interesting in running AVI, MPEG, or QuickTime movies, you must also have Microsoft Internet Explorer version 4.0 or higher, installed on your computer.

2.2 Single-computer Cortex without graphics (\REMOTE32\SGL32.EXE)

The SGL32.EXE version is a 32-bit DOS program, which does not display graphics. The serial communications functions and the analog/digital capabilites are supported. Therefore, this version can be used when graphics are not necessary, or when the graphics are displayed by some other means. The advantage of this version is that it is built with 32-bit DOS, so the amount of memory that can be used is not limited to the 640 KB boundary.

2.2.1 Computer

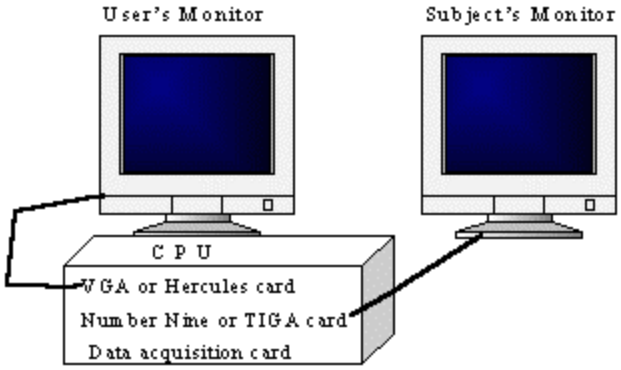
The single-computer Cortex computer controls experimental timing, data collection, and can send serial commands to another computer. The Cortex computer contains the data acquisition card.

A Pentium computer (at least 100 MHz) with 32 MB RAM and at least a 1 GB hard drive for data storage is recommended. These specifications are based on the assumption that the computer will have Win95/Win98 installed on it also. Similar to the other DOS versions of Cortex, the single-computer 32-bit Cortex program must be run from a computer which is booted up in true DOS.

2.2.2 Data acquisition cards

All of the data acquisition boards originally specified for the other versions of Cortex should also work with the single 32-bit version of Cortex. See section 2.1.5 above.

2.3 Single-computer Cortex with graphics (NNIOS and NUM9GXI)



2.3.1 Computer

The basic computer must be at least an AT class machine (80286 or better) with 640K memory, keyboard, and mouse. A VGA video board (or compatible video card, such as a Hercules monochrome board) is needed in addition to a compatible monitor.

2.3.2 Graphics

For visual stimulation studies, a Pepper SGT-plus graphics card with at least 1 MB of memory, or a #9GXi-TC graphics card with 4megs VRAM and at least 1 meg DRAM, is needed for the subject's visual display. The Pepper SGT will work with any VGA or multiscan monitor and the #9GXi-TC can support multiple resolutions and scan rates. Many labs use two subject monitors (one for the subject and one for the user....they may be in separate rooms) and use a video splitter such as the VOPEX-2V-H from NTI). Please note that the Pepper SGT-plus and the #9GXi-TC graphics cards are no longer manufactured. In fact, the CORTEX development team does not have these graphics cards, so this version of CORTEX has not been tested for the NNIOS and NUM9GXI executables.

2.3.3 Data acquisition

All of the data acquisition boards originally specified for the other versions of Cortex should also work with the NNIOS and NUM9GXI versions of Cortex. See section 2.1.5 above.

2.4 Useful Web Sites for Finding the Necessary Hardware and Software

BlackBox Corp.	http://www.blackbox.com/
Cortex home page	http://www.cortex.salk.edu/
ComputerBoards, Inc.	http://www.computerboards.com
Keithley/Metrabyte	http://www.keithley.com/
Scitech Software	http://www.scitechsoft.com/
Microsoft DirectX	http://www.microsoft.com/directx/homeuser/downloads/default.asp
NTI	http://www.networktechinc.com/
Microtouch	http://www.3m.com/3mtouchsystems/

3. Installation

3.1 Data Acquisition Board Setup

3.1.1 Typical Keithley/Metrabyte DASH-16 settings

- Base Address: 0x300 or 0x310
- DMA Channel: By default, the board requires DMA channel #1. If something else is using that DMA channel, there is an onboard jumper that can be switched to use DMA channel #3. Otherwise, use the Windows Control Panel to move the conflicting device to a different setting.
- should be set for bipolar, 16 analog inputs
- gain = whatever is convenient
- analog eye inputs (EOG): analog input channels 3 High and 4 High
- evoked potentials (EPP): analog input channels 1 High and 2 High
- bar contact input: digital input 2 and 3
- solenoid reward output: digital output 1. Cortex outputs a brief TTL pulse to trigger a solenoid control circuit. By default, the duration of the pulse is 20 ms. It can be configured to different times via the Cortex Run:Parameter:General menu options, or using an external variable (ms_reward_duration) in the timing file.
- spike inputs (2 or more separate spike channels): digital inputs 0 and 1 (for the first two inputs), which are fed by flip-flops (provided by the user's electronics shop...see Section 3.3 entitled "Data Acquisition Interface Setup"). Typically one uses the "user 1 and 2" lugs on the Metrabyte interface connector board to access the flip-flops.
- output to clear flip-flops: digital output 0

3.1.2 Typical PIO24 or CIO-DIO24 settings

The PIO24 board is shipped with a base address of 0x300. The base address must be changed with onboard jumpers, if this address is already occupied. By default, Cortex uses Port A as 8 lines of input, Port B as 8 lines of output, Port C (0-3) as 4 lines of input, and Port C (4-7) as 4 lines of output. This is configured at startup in Cortex, but you can dynamically change this in your timing file to give a different distribution of inputs and outputs.

3.1.3 Typical Computerboard CIO AD-16, CIO-DAS16/F, and CIO-DAS1602/12 settings

The Computerboard CIO AD-16, CIO-DAS16/F and CIO-DAS1602/12 can be considered to be a combination of a DASH-16 board and a PIO24 digital I/O board, with the address space of the PIO24 following that of the DASH-16. Because of the PIO24 on board, it is usually necessary to use the Computerboard with a base address of 0x300 for the DASH-16 component rather than 0x310. If the DASH-16 component has an address of 0x300, the PIO24 component can be set with an address of 0x310, which is a safe address. Otherwise, if you set the base address for the DASH-16 component at 0x310, the address space of the PIO24 is automatically set at 0x320, which apparently can conflict with other devices on the bus. Note: The CIO-DAS1602/12 board is shipped with the analog inputs set to UNIPOLAR. Therefore, the switch on the board must be changed to the BIPOLAR setting in order for Cortex to collect analog data properly.

3.1.4 PCI-DAS1602/12 settings

The PCI-DAS1602/12 also contains a DASH-16 portion and a PIO24 portion. It is a plug-and-play device, which does not have any onboard jumpers to configure the base addresses. Every time the computer boots up, the BIOS assigns base addresses to the board which will not conflict with the other devices in the computer. It is possible that each time that the computer boots up, the board will have different addresses. Also, instead of having just one base address, the PCI-DAS1602 has five base addresses. Because of this complicated situation, Cortex will just query the BIOS and board internally, to figure out these five addresses. The base address that is

supplied on the DEVICE PCIDAS1602 line in the Cortex.cfg file will be ignored, even though it is a required parameter. For other technical notes about the PCI-DAS1602/12, refer to Appendix B.

3.1.5 PCI-DIO24 settings

The PCI-DIO24 is the PCI version of the PIO24 board. It is a plug-and-play device, which does not have any onboard jumpers to configure the base addresses. Every time the computer boots up, the BIOS assigns base addresses to the board which will not conflict with the other devices in the computer. It is possible that each time that the computer boots up, the board will have different addresses. Also, instead of having just one base address, the PCI-DIO24 has two base addresses. Cortex will query the BIOS and board internally, to figure out these two addresses. The base address that is supplied on the DEVICE PCIDIO24 line in the Cortex.cfg file will be ignored, even though it is a required parameter.

3.2 Graphics Board Setup

3.2.1 Pepper SGT settings

- 640x480 resolution (default)
- Should be set for no emulation
- Put the following line in your config.sys file
- device=nnios.sys /m=A000:64;
- (Note: If the /m=A000:64 option generates a "memory windows" error, do not put it into the device line.)
- Copy the nnios.sys file into the root directory of your computer.
- Shut down the computer and install the Sgt. Pepper card into the computer.
- Restart the computer. When your computer boots up, it should tell you that it has loaded the Pepper SGT device driver successfully.

3.2.2 #9GXi-TC settings

Set the DIP switches for the board as listed below.

DIP Switch Settings:

Sw1 Sw2 Sw3 I/O Address
On on on 280-28F
On on off 290-29F
On off on 2A0-2AF
On off off 2B0-2BF
Off on on 2C0-2CF
Off on off 2D0-2Df
Off off on 2E0-2EF
Off off off Reserved

Sw4 Reserved
On

Sw5 Enable VGA
On On-board VGA enabled
Off On-board VGA disabled

Sw6 VGA BIOS Transfer mode
On Word (16-bit) mode
Off Byte (8-bit) mode

Sw7 Monochrome emulation

Off Monochrome enabled
On Monochrome disabled

Sw8 Reserved
On

- Set the first three switches to the desired address, and then set ON, Off, On, Off, On (assuming you don't plan to use the on-board VGA).

- Once the board is physically installed, run the install program from the floppy disk from a DOS prompt.

- Once that is done, replace the TIGA communication driver in the autoexec.bat with one of the drivers in the \util_bin directory of the CORTEX download. That is, replace the line "c:\tiga2\tigacd.exe" in the autoexec.bat file with c:\cortex\util_bin\ticortcd.exe. Upon bootup, if you get an error which says something like "#9GXI Comm buffer failure..", then replace ticortcd.exe in the autoexec.bat file with one of the other executables in the \util_bin directory (i.e., t2cortcd.exe, or tigacd0.exe).

- After that, you can set various aspects of the board (i.e. default resolution, screen offsets, and flicker rates) for each resolution in the gxi.bat utility in the c:\tiga2 directory.

3.2.3 Graphics card setup for Scitech Display Doctor and DirectX Receive Programs

Install the graphics card according to the graphics card manufacturer's instructions. Install the SDD or DirectX driver. These drivers should recognize the graphics card, if the driver supports it.

3.3 Data Acquisition Interface Setup

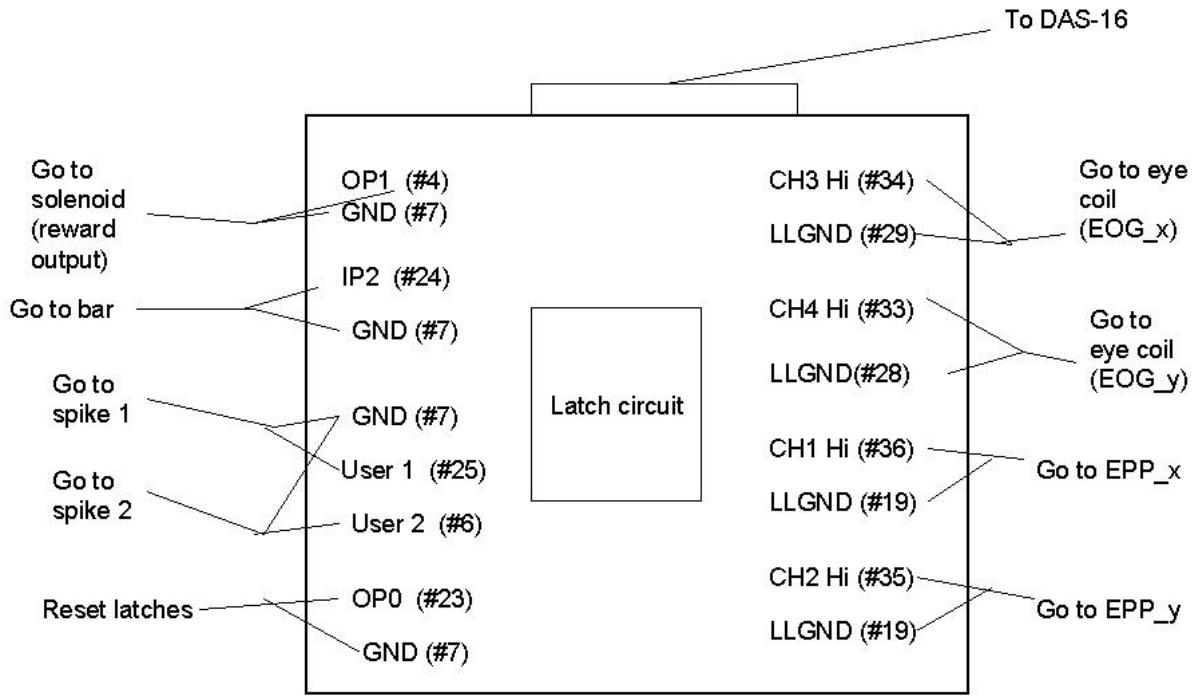
3.3.1 Spike Flip Flop Circuitry

The Dash-16 does not latch digital inputs (for purposes of spike acquisition, neither does the PIO24). This seems to be typical for digital inputs boards for the PC, except possibly for a handshaking line. So if you want to collect neuronal spikes, you must add flip-flops between the TTL pulse of your window discriminator (a +5v digital pulse) and the digital input ports of the board. Otherwise, if your TTL pulse is less than 1 msec long, it may be "gone" by the time the Cortex program gets around to examining the port; if longer than 1 msec, Cortex will read more than one spike for each pulse. A circuit diagram for the flip-flops can be obtained from the Cortex web site. The flip-flops are cleared by digital output #0 (courtesy of the software and the flip-flop circuit). The spike inputs go to user inputs 0 and 1. Behavioral inputs go to digital inputs 2 and 3 (bar up/down = 2, bar left = 2, bar right =3), which naturally are not latched.

3.3.2 Thalamus

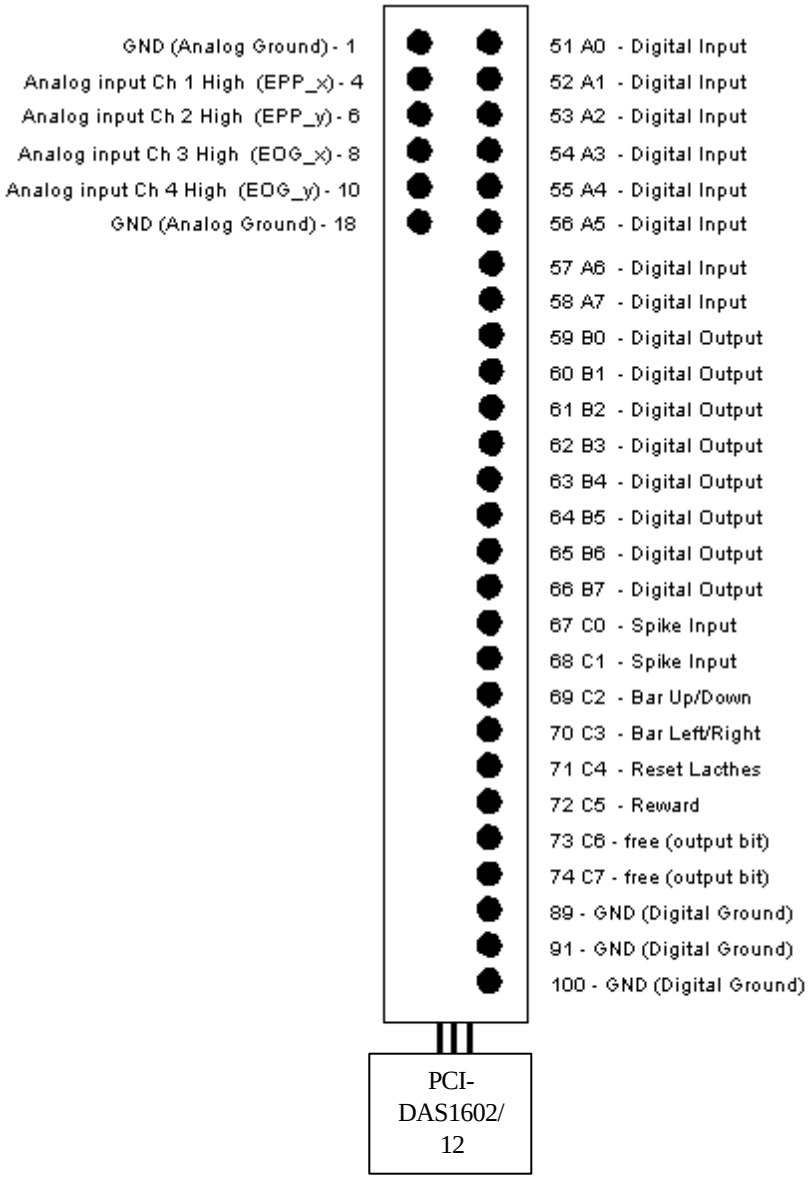
Although Cortex does not require any interface beyond the screw-terminal box or panel sold by Metrabyte or Compuboard (plus a couple of flip-flops for the spikes), Andy Mitz developed a fabulous general-purpose interface called Thalamus. Thalamus is an interface and interconnect box that goes between the computer I/O boards and the outside world. It contains BNC connectors for up to 8 A/D channels and up to 8 spike inputs. It has input latches for all 8 spike inputs. Four of the 8 A/D inputs can have built-in anti-aliasing filters. Thalamus also has "D" type connectors for digital input and output. There are 8 "touch pad" circuits in the Thalamus unit to aid with touch switches or to condition other switches. There is also a one-shot and driver for delivering rewards. Thalamus was developed by the Laboratory of Neurophysiology, NIMH. Drawings for the Thalamus are available on the Cortex web site and at the Laboratory for Neurophysiology's web site.

3.3.3 Screw Terminal Interface Diagram for the DASH-16, CIO-DAS16/F, CIO-AD16, and CIO-DAS1602/12

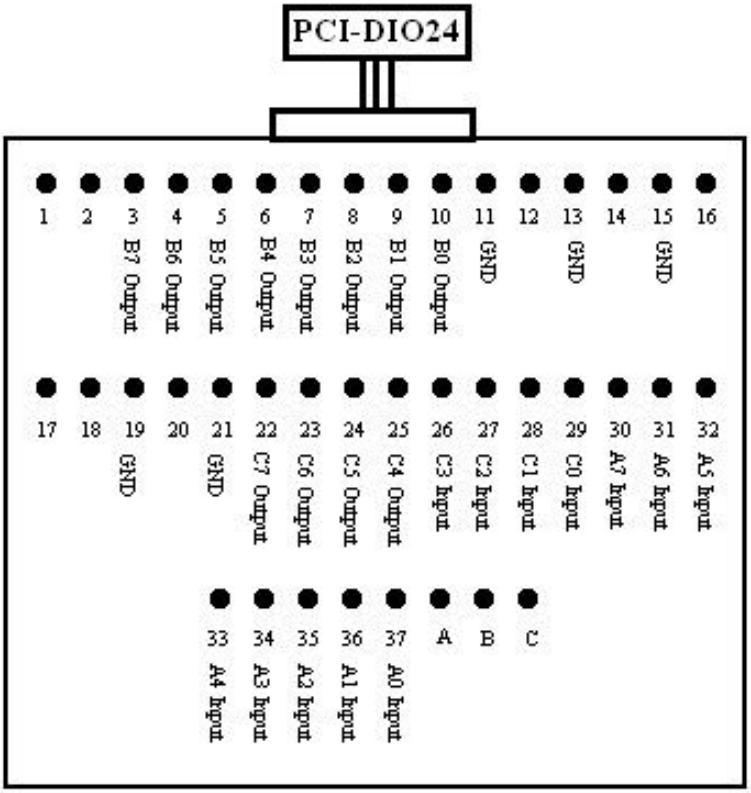


(#) Represents pin number
Resetting of latches is usually implemented in the latch circuitry.

3.3.4 Screw Terminal Interface Diagram for the PCI-DAS1602/12 board



3.3.5 Screw Terminal Interface Diagram for the PCI-DIO24 and CIO-DIO24 boards



- If you are using the PCI-DIO24 or CIO-DIO24 board with the PCI-DAS1602/12 board you will use the Reset line on the PCI-DAS1602/12 board to latch the spikes.
- If you are using the PCI-DIO24 or CIO-DIO24 board alone you will use Pin 25 as the Reset line.
- In order to get the reward and bar to work properly with the PCI-DIO24 or CIO-DIO24 board alone, you must use the DEVinp and DEVoutp functions in your timing file.
- The above diagram shows the default port settings, but the board can be programmed to have different input and output lines using DEVoutp.

3.4 Sound Card Setup

The dual computer Cortex program has the ability to play sounds. Sound can be used not only as stimuli, but also as an auditory feedback mechanism to assist in training. Up to 256 different sound (.wav) files can be played during an experiment. Additionally, the volume and mix of the sound from the speakers can be changed during the experiment through function calls in the Cortex timing file. See the online Function Reference Manual at http://www.cortex.salk.edu/CortexFunctionReference.php.

3.4.1 Sound for DOS Scitech Display Doctor (rvesactx.exe) version

A SoundBlaster card must be installed in the receive computer. Any modern SoundBlaster card should work (e.g., SB16, AWE32, or AWE64). It might also work on SB16 compatible cards, but it has not been tested. If the SoundBlaster card has been installed properly in the computer, there is no additional configuration necessary to use the Cortex sound capabilities. The code reads the DOS "BLASTER" environment variable to get the addresses and IRQs of the card. Certain problems with the sound initialization and execution during the use of Cortex are logged to the file "rvesactx.err" on the receive computer. Please note that the sound (.wav) files must be present on the receive computer.

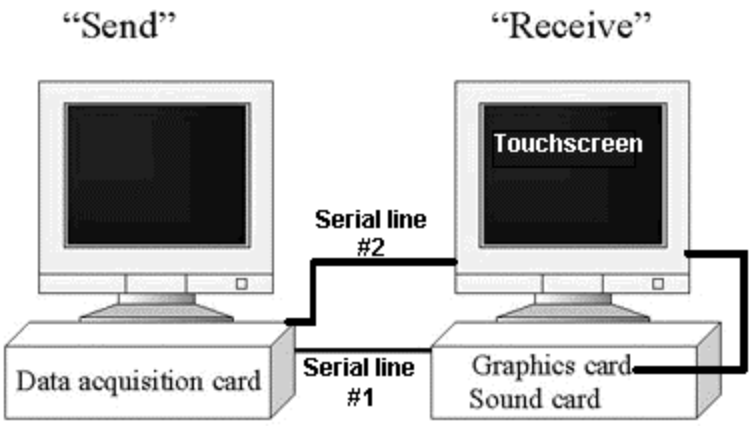
3.4.2 Sound for DirectX (dxrecv.exe) version

Any sound card which is supported by DirectX can be installed in the receive computer. After the sound card is installed in the computer, the DirectX driver must be installed. Certain problems with the sound initialization and execution during the use of Cortex are logged to the file "dxrecv.err" on the receive computer. Please note that the sound (.wav) files must be present on the receive computer.

3.5 Touchscreen Setup

Prior to Cortex version 5.7 release 10, a touchscreen could only function properly on a single computer Cortex setup. For release 5.7 release 10 (and later), the dual computer version of Cortex also works properly with a touchscreen. The code was written for and tested on a Microtouch touchscreen. Please refer to Section 3.6.4 of this document for the basic dual computer installation instructions. If you have never set up a dual computer version of Cortex before, you may want to set up a basic system first before attempting to use the touchscreen.

3.5.1 Hardware Setup



On the receive computer:

- Connect the Microtouch touchscreen graphics cable to the video card in the receive computer.
- Connect COM1 on the receive computer to COM1 on the send computer via serial cable (for Cortex communication).

On the send computer.

- Connect the Microtouch touchscreen controller serial cable to COM2 on the send computer.
- Connect COM1 on the send computer to COM1 on the receive computer via serial cable (for Cortex communication).
- Install Microtouch Touchware DOS driver software (see section 3.5.2 below).

(Note: The above instructions were simplified by assuming that COM1 is used for the Cortex communications port.)

3.5.2 Microtouch Touchware Setup

The latest Microtouch Touchware driver software is available from the Microtouch web site (http://www.3m.com/3MtouchSystems/downloads/legacy.jhtml). You must install the DOS version of the Touchware driver.

1. Reboot the send computer into DOS, since Touchware must install and run from DOS. During Touchware installation, the driver tries to find the touchscreen controller. If it does not find the controller, run microcal.exe. If microcal.exe finds the controller, write down the settings for the communications port number, the IRQ number, and the baud rate. Edit the file DOSTOUCH.INI, and replace the current values with the ones reported by microcal. You should now be able to run the Touchware driver (DOSTOUCH.EXE), and it should report that the driver was installed.
2. Calibrate the monitor using microcal.exe on the send computer. Be aware that the calibration screen will appear on the send computer's display...not the touchscreen! However, if you touch the corners of the touchscreen, it should be detected properly. For a more accurate calibration, you can temporarily attach the touchscreen video cable to the video card of your send computer (instead of your normal send computer monitor). This will allow you to see the calibration markers on the touchscreen itself while running microcal.exe.
3. To work properly with Cortex, it is suggested that you edit the DOSTOUCH.INI to contain the following values:
   - DefaultVirtualSize=640 480 (This allows the touchscreen to map the touch from (0,0)(999,999) to the required (0,0) (640, 480).)
   - CursorOffset=0 (so that when the screen is touched, no value is added to the location)
4. You may also want to experiment with the TouchMode value in the DOSTOUCH.INI file. These values can be changed by running the DOSPANEL.EXE program.
5. Before running Cortex, you must always boot up your computer into DOS and run the DOSTOUCH.EXE driver on the send computer.

3.5.3 Cortex Configuration File Setup for Touchscreen Operation

In order to specify to Cortex that the touchscreen will be used, the following line must be included in the CORTEX.CFG file. (Refer to Section 4.6 for general information about the Cortex.cfg file.)

```
TOUCH_SCREEN <int ms_per_tik > [<store_touch_data]
```

- For example, the line could be:

```
TOUCH_SCREEN 50
```

- where 50 is the number of milliseconds between queries of the touchscreen
- (To emulate the touch screen using the mouse, use a negative number for the ms_per_tik value.)

Since the calculation of the location of the touch is based upon the "pixels-per-dva" setting in the CORTEX.CFG file, it is imperative that the values are set correctly in the GRAPHICS_SPECS line.

In Cortex version 5.9.3 and higher, the optional parameter, <store_touch_data, was added to the TOUCH_SCREEN parameter of CORTEX.CFG. This parameter allows the user to specify whether or not the touch values should be stored to the EPPbuf, the CODE/ISI buf, both places, or neither place.

- Valid values are:
  EPPBUF
  CODEBUF

NEITHER
BOTH (default)

- If no value is provided, the touch data will be stored to both the EPPbuf and CODE/ISI buf.

3.6 Cortex Software Setup and Program Execution

In all configurations, the first step is to download the latest Cortex distribution file from the Cortex web site. Using WinZip or PKUNZIP.EXE, unzip the distribution file.

3.6.1 SGT Pepper version

The SGT Pepper version of Cortex will automatically be installed into the \SGTPEPPR directory on your computer.

You will need to edit the cortex.cfg file in that directory to reflect the settings for your system.

You run the SGT Pepper version of Cortex by entering **nncortex -n**, where '**n**' is the name of the disk drive where you would like temporary files stored. This drive can be a RAM disk, to speed up performance. The temporary files do not contain the experimental data but rather hold information used internally by Cortex, such as binary representations of the items. Another command line argument that can be used is the filename of a "Save" file (created through the Cortex menus with Disk:Save). If a filename is encountered on the command line, Cortex interprets it as the name of a Save file, and automatically loads those settings into Cortex.

3.6.2 TIGA (#9GXi-TC) version

The TIGA version of Cortex will automatically be installed into the \NUM9GXI directory on your computer.

You will need to edit the cortex.cfg file in that directory to reflect the settings for your system.

You run the TIGA version of Cortex by entering **ticortex -n**, where '**n**' is the name of the disk drive where you would like temporary files stored. This drive can be a RAM disk, to speed up performance. The temporary files do not contain the experimental data but rather hold information used internally by Cortex, such as binary representations of the items. Another command line argument that can be used is the filename of a "Save" file (created through the Cortex menus with Disk:Save). If a filename is encountered on the command line, Cortex interprets it as the name of a Save file, and automatically loads those settings into Cortex.

3.6.3 Single-computer with no graphics (SGL32) version

The SGL32 version of Cortex will automatically be installed into the \REMOTE32 directory on your computer.

You will need to edit the cortex.cfg file in that directory to reflect the settings for your system.

You run the SGL32 version of Cortex by entering **sgl32 -n**, where '**n**' is the name of the disk drive where you would like temporary files stored. This drive can be a RAM disk, to speed up performance. The temporary files do not contain the experimental data but rather hold information used internally by Cortex, such as binary representations of the items. Another command line argument that can be used is the filename of a "Save" file (created through the Cortex menus with Disk:Save). If a filename is encountered on the command line, Cortex interprets it as the name of a Save file, and automatically loads those settings into Cortex.

3.6.4 Dual Computer Cortex Software Installation Instructions

The Cortex "send" computer controls experimental timing, data collection, and sending commands to the graphics ("receive") computer. The two computers are connected via a serial cable between COM ports.

By default, Cortex uses COM1 as the communications port and 115200 as the baud rate on each computer. However, COM2 can also be used for Cortex communication. The send side can be changed to use COM2 and a different baud rate in the Cortex.cfg file with the COM_PORT line. The receive program must be changed with

command line arguments. That is, if you wanted to use COM2 and baud rate 9600 on the receive computer, you would have to supply COM2 and @9600 as command line arguments.

There are two versions of the "receive" program: a DOS Scitech Display Doctor based version, and a Windows DirectX based version. You only need to set up and use one of these receive programs!

3.6.4.1 Scitech Display Doctor "Receive" (RVESACTX.EXE) software installation

- Install SciTech Display Doctor. A free copy can be downloaded from the SciTech web page (http://www.scitechsoft.com/products/ent/free_titles.html).
- During the installation of Display Doctor, allow the autoexec.bat to be changed. After installation, the autoexec.bat must contain the line "C:\SDD\UNIVBE32.EXE -w", so that the Display Doctor will automatically be launched upon bootup. Also, make sure that the linear frame buffer capability is enabled in Display Doctor, by double-clicking on the Display Doctor Control Center icon and looking at the "Compatibility Testing" section.
- Download and unzip the latest Cortex distribution code from the Cortex web page. The minimum files that the receive computer must contain are: RVESACTX.EXE, DOS4GW.EXE, and PC8X8.FNT. These three files must all reside in the same directory.
- If you have bitmaps associated with your tests, they must also reside on the receive computer, since the bitmap files do not get transmitted over the serial line. (If they are not there, you will get the error "can't calc image size" on the send computer.)
- Boot up the computer into true DOS (see section 3.6.5).
- To run the Scitech Display Doctor based receive program, change to the Cortex directory containing the executables (usually \REMOTE32). In DOS, type "RVESACTX" to start the receive program running.

3.6.4.2 DirectX Receive Program (DXRECV.EXE) software installation

- Install the DirectX driver. In order to use the DirectX receive program (DXRECV.EXE), the graphics card in your receive computer must be supported by DirectX, and the DirectX 6.0 (or higher) driver must be loaded. If these criteria are not met, the DXRECV.EXE program may hang, and the WCSEND.EXE program (on the send computer) will report the error “Couldn’t initialize graphics board” upon startup. If this happens, you must install the DirectX driver on the receive computer. The DirectX driver is freely available from the Microsoft web site. Go to the web page http://www.microsoft.com/directx/homeuser/downloads/default.asp. Download and install the current version of DirectX 7.0. If you are only interested in running DXRECV.EXE, you only need to download the DirectX driver. (If you are interested in changing and recompiling the DXRECV code, you must download the DirectX SDK. See Section 8.3.)
- If you are interesting in running AVI, MPEG, or QuickTime movies in DXRECV.EXE, you must also have Microsoft Internet Explorer version 4.0 or higher, installed on your computer.
- Download and unzip the latest Cortex distribution code from the Cortex web page. At a minimum, the \Wcortex directory must reside on the receive computer.
- If you have bitmaps associated with your tests, they must also reside on the receive computer, since the bitmap files do not get transmitted over the serial line. (If they are not there, you will get the error "can't calc image size" on the send computer.)
- The DirectX receive program (DXRECV.EXE) is a Windows application, which must be run from Windows (not DOS). The easiest way to run it, is to make a shortcut on the desktop of the receive computer to the executable, \WCORTEX\DXRECV.EXE. When you want to run the program, just double-click on the DXRECV.EXE shortcut icon. An hourglass may (or may not) appear for a few seconds. When you start wcsend.exe on the send computer, the receive computer display should turn black.

NOTES:

- If you are running Dxrecv.exe in Windows 2000 or Windows XP, you may find that the Display Properties settings can not be set to 640 x 480. Therefore, you must change the shortcut properties of DXrecv.exe. This can be achieved by right-clicking on the shortcut and selecting Properties. Once the Properties sheet comes up, click the Compatibility tab. On this tab you will see a "Compatibility mode" group box. Select “Run this program in compatibility mode for:”. Once this box is checked, you

will be able to choose an operating system from the drop down list box. Choose Win95 or Win98/Me, depending on your system's Send computer. If you are running Cortex at 8 bits per pixel color, it is also recommended that you check the "Run in 256 colors". Also, if you are running at 640 x 480 resolution, you must check "Run in 640x480 screen resolution." These two options are located in the "Display settings" section on the Compatibility tab panel.

- If the user would like to change the default baud rate and COM port for the DirectX receive program (DXRecv.exe), the user must change the command line arguments in the Properties page of the Dxrecv.exe shortcut on the desktop. The parameters should be added onto the "Target:" line. For example, the "Target:" line would read:

```
c:\cortex\wcortex\dxrecv.exe COM2 @115200
```

- Notice that there are no spaces between the COM and the 2, and no spaces between the @ and the baudrate. If these command line arguments are omitted, the default values are a comport setting of COM1 and a baudrate of 115200. For Win2000 users, the shortcut must be created from the directory \dxsource\dxrecv\release\dxrecv.exe for the command line arguments to work properly. If this is done, then the working directory of the dxrecv.exe program will be \dxsource\dxrecv\release, not \wcortex.

3.6.4.3 "Send" (WCSEND.EXE) software installation

- It is not necessary to install any special graphics driver on the send computer. Therefore, just download and unzip the latest Cortex distribution code from the Cortex web site.
- Edit the c:\cortex\remote32\cortex.cfg file to reflect your system setup. Make sure that the line "COM_PORT 1 115200" has been uncommented.
- The testing files (items, conditions, timing files) must be located on the send computer. (The associated *.lut files must be located in the specified directory on the send computer. If a path to the .lut file is not specified in the conditions file, Cortex will look for the .lut file in the directory from which the send program is running.)
- Boot up the computer into true DOS (see section 3.6.5).
- To run the send program, change to the Cortex directory containing the executables (usually \REMOTE32). In DOS, type "WCSEND" to start the send program running.
- For the DOS version of send and receive, the serial connection between the two computers can be broken while Cortex is running by the keyboard combination CTRL-ALT-SHIFT. You might have to use this keyboard combination several times, and hit the ESC key. To set the keyboard sequence to SHIFT-CAPSLOCK-ALT instead of CTRL-ALT-SHIFT, set the environment variable "CORTEX_ESC_OPT" equal to 1 before running Cortex. If the environment variable is not set, or if it set to 0, then the default CTRL-ALT-SHIFT key sequence will be used.

3.6.5 Booting up the computer in "true" DOS

To run in true DOS mode, shut down your computer and choose "Restart the computer?". When the computer displays the "Starting Windows 95…." message, press the F8 key. In the menu that appears, choose the option for "Command Prompt Only". The computer will boot to a C:> prompt.

In order to make the computer boot up in DOS without having to hit F8, you can do the following:

1) For Win95/98, in C:\ there is a hidden system file called MSDOS.SYS. In order to be able to edit it, go to a DOS prompt, change to C:\, and then type "attrib -h -r -s msdos.sys".
2) Of course, before you edit it, make a backup copy!!
3) Edit msdos.sys. In the [Options] section, add the following lines (or change them if they are already there) to look like this (explained below):

   BootMenu=1
   BootMenuDefault=6
   BootMenuDelay=60
4) Save the file, and change it's attributes back to hidden, read-only, system file, by typing "attrib +h +r +s msdos.sys".
5) Reboot, and it should show you the menu.

NOTE: The above options are set so that:

a) BootMenu=1 means show the menu. If you no longer want to see the menu, you will have to edit it to say BootMenu=0.
b) BootMenuDefault=6 means that if you don't press any menu key, it will automatically boot up with option 6 (command prompt only). In Win98, the value should be 5.
c) BootMenuDelay=60 means that it will wait 60 seconds before running the default menu item. The default time is 30 seconds. Of course, you can make it less, if you want.

3.7 Uninstalling Cortex

Cortex does not place any information in config.sys, autoexec.bat, Windows system directories, etc. Cortex files will only exist in directories where you have copied the files. When installing a new version, it is best to just create a new directory. Delete the old directory when the new version works with your experiment. Remember to change the Cortex.cfg file to be appropriate for your setup.

4. Using Cortex

A fundamental feature of Cortex is that it is organized around "trials". That is, data are collected in discrete trials that last anywhere from a few milliseconds to a few minutes, followed by an "intertrial" interval in which the program does bookkeeping such as storing data from the previous trial and setting up the conditions for the next trial. The duration of the intertrial interval can be increased by the user, but the minimum may be about half a second or so (or longer if you need to read in large images from disk files).

Before Cortex begins running trials, the user must specify the different varieties of experimental conditions that are to be run in the session.

4.1 Setting up the experimental conditions.

To set up the experimental conditions, it is necessary to write at least three ASCII text files "off-line", which you then import into Cortex. These three files can be written using your favorite text editor. Word processing programs cannot be used unless they can output a standard ASCII text file, which some word processing programs have as an option. It is best to work from an already existing set of files (such as the examples and demos provided with Cortex), which you can edit and modify, rather than to try to write the files from scratch the first time.

The three files you must write are **items**, **conditions**, and **timing** (state system) files. For a given experimental session, you can import only a single item and condition file into Cortex, but multiple timing files.

For the items and conditions files, you will note that at the top of each file, there are headings such as ITEM, TYPE, CENTERX, etc. These headings are special keywords that you should not alter. The first and last character of each keyword define the beginning and end of the data field columns located under the keywords. That is, you enter your specifications in the columns situated under each heading word, taking care not to extend any data beyond the first or last character of the heading word. Because "tabs" can mean any number of spaces, depending on your text editor, Cortex cannot interpret tabs in the files. Thus, DO NOT USE TABS. USE ONLY SPACES between values in the items and conditions files. If you use tabs, previous versions of Cortex would read in your file in a jumbled manner and become very confused. The recent versions at least warn you that tabs are present in your file. If you get a warning about tabs, re-edit your file to remove them and then reimport the file. The state system file is free-format, so tabs are fine. The format of the files is given next.

4.2 Item files

For experiments in vision, you probably have in mind a number of visual stimuli that you want to present to a particular neuron or subject. Before specifying how and when you want the stimuli to be presented, it is first necessary to describe to Cortex the stimuli you want to present. These stimuli are referred to as ITEMS in Cortex.

You must describe each item you want to use in a given experimental session in the ITEMS ASCII file, which you import into Cortex at the start of the session. For example, you might want to compare the responses of a neuron to 20 bars of light that vary in color and orientation. In the items ASCII file, you would describe the 20 items in the format required by Cortex, including the type of visual stimulus (in this case, bars of light), their height, width, orientation, color, and position relative to some reference point on the display screen (to be described later).

An example items file is shown below:

ITEM	TYPE	FILLED	CENTERX	CENTERY	BITPAN	WIN_WIDE	WIN_TALL	HEIGHT	WIDTH	ANGLE	INNER	OUTER	-R-	-G-	-B-	C	----FILENAME----	
-4	1	1	0.00	0.00	0			1.00	1.00	0.00			0	0	0	x		
-3	1	1	0.00	0.00	0			0.30	0.30	0.00			255	255	255	x		
-2	1	1	0.00	0.00	0			1.00	1.00	0.00			255	255	255	x		
-1	1	1	0.00	0.00	0			1.00	1.00	0.00			255	255	255	x		
0	1	0	0.00	0.00	0			3.00	1.00	0.00			50	50	50	x		
1	1	0	0.00	0.00	1			3.00	1.00	45.00			100	100	100	x		
2	8		0	0.00	0.00	2	1.8	1.8							230	0	230	x
face.ctx																		
3	1	1	0.00	0.00	1			3.00	1.00	45.00			250	250	250	x		

Some of the columns you must fill in within the items file are as follows:

ITEM. The ITEM refers to the item number, which you henceforth use in Cortex as the name or identifier of the stimulus you have created. Some items have negative numbers. These are "special" items that are used internally by cortex. Their meaning is usually as follows, but can be used differently:

-4 Background item
-3 Fixation spot item
-2 Reference field
-1 Reference point
0 Play item

TYPE. Cortex can handle many different kinds of items, which generally fall into two classes. The first class Cortex "knows" how to create itself. This class includes bars, circles, ellipses, annuli, and ASCII characters. Each of these types has an associated code number, which you enter in this column (see below for a list of type codes). In addition, you must enter the appropriate information about the items in the other relevant columns in the file. For example, bars need a length and width, circles need a radius, etc. The second class is bitmaps and movies, which are items that you have created outside of Cortex and saved in a disk file. These also have a "type" number, which you enter in this column. For these bitmap and movie items, you must also give the name of the disk file that holds them in the FILENAME column. For the format of these bitmapped files, see Section 7. All items, regardless of type also need an CENTERX and a CENTERY, which you enter in the appropriate column.

Item type codes:

```
1      bar
2      circle
3      circular annulus  (outer and inner diameter should be entered in the inner and outer columns)
4-6    reserved
7      ascii character
8      bitmap  (name of file must be in FILENAME column)
9      ellipse
10     annular ellipse
11      movie     (name of file must be in FILENAME column)
12     regular polygon
13     ascii string
14     cross
```

FILLED. Objects such as bars, circles and annuli can either be filled or just an outline. If you put a '1' in this column the object will be filled, and if you put a '0' it will be an outline. If a FILLED parameter is not specified, the item will be filled by default.

CENTERX, CENTERY. Each object must have a location on the screen. This sets the x and y axis location, in degrees, from the reference point. The reference point is the location that Cortex defines to be the center of its coordinate system for placing items on the screen. The user can configure this location.

BITPAN. "Bitpannable" means that you plan to pan objects around inside a stationary window on the screen. A '0' in this field means that the object is not bitpannable, and cortex should treat it like any other object. A '1' in this field means that it is bitpannable, that cortex should utilize the window size you specify (see below), and that cortex should replicate the object 4 times, to give you more object to pan around with. A '2' in this field means the same as 1, but Cortex won't replicate it. If you use option 1, there may be sharp transitions in the image at the boundaries of the four replications, but for some purposes, this is OK. Also, you may need to set the bitpan field to 2 if you have an extremely large object. The Sgt. Pepper card will not allow you to import a bitmap larger than the display screen (640 x 480), unless you specify a window around it that is smaller than 640 x 480.

WIN_WIDE, WIN_TALL. When you give a window length and width (in degrees), and the object is bitpannable, Cortex will create a window around your item. This will allow you to pan a large object around inside a smaller window. This is only available when bitpan=2 for Manual.

HEIGHT, WIDTH, ANGLE. Bars require a length, width and orientation, in degrees.

OUTER. Circles and ellipses require an outer diameter.

INNER, OUTER. Annuli require an inner and outer radius.

C (Color), -R-, -G-, -B-. For items that Cortex knows how to create, such as bars and circles, you must tell Cortex what color they should be. In the color column you may, if you wish, give a letter representing the color name you want. 'R', for example, stands for red. However, these default colors will probably not have the exact red, green, and blue values you want, so you should really list the actual rgb values you want in the file instead of just giving a color name. To give the actual rgb values, use 'X' as the color name in the color column, and then enter the appropriate values in the -R-, -G-, and -B- columns. These values range from 0 (off) to 255 (maximal brightness) for each red, green and blue gun on the CRT.

For items that are stored as bitmaps, the color you list in the items file is ignored. The colors are determined by the color lookup table, which is described in Section 7.3. The name of a color lookup table file can be entered in a menu within Cortex or listed in the conditions file (see below).

FILENAME. For bitmap and movie items, you must give the filename. The name can include a path, if it is not in the current directory. Note, however, that the length of the name, including the path cannot extend past the ends of the ------FILENAME------ column (i.e., 20 characters total).

4.3 Conditions files

The second ASCII file you must import into Cortex is the conditions file. Cortex can interleave trials from a large number of experimental conditions within a single experimental session. Each condition determines what happens on a particular type of trial. In this file, you specify which items are linked together within each condition, and which timing structure is associated with that condition.

An example of a conditions file is as follows:

```
  COND# TEST0 TEST1 TEST2 TEST3 TEST4 TEST5 TEST6 TEST7 TEST8 TEST9 BCKGND  TIMING TRIAL_TYPE  FIX_ID  ---COLOR-PALETTE---
    1    2             2                                             1       2      0           -3      lookup.lut
    2    3     6       7     5     4                                 1       1      0           -3
    3    4     4                                                     1       2      0           -3      lookup.lut
    4    6     7             8     9                                         1      1           0       -3
    5    10    9             3     5                                         1      1           0       -3
    6    12    13    12      13                                      1       3      0           -3
    7    13    12    13      12                                      1       3      0           -3
```

COND# The condition number is, as you might expect, the number of each experimental condition. Each line typically defines a condition, but if you need to list more than two items on a given test screen, you can list the additional items on the next line. In the above example, condition number 2 extends across two lines. Depending on the version of Cortex you are using, the maximum number of items per test screen is either 4, 8, or 16.

TEST0, TEST1, etc. For each test screen you list the items that you want to appear together. The item numbers refer the items listed in the items file. On a single trial, you can have up to ten successive screens worth of images presented. The timing and sequence of the display screens is given in the state system. For example, the state file could be programmed so that in condition 2, TEST0 containing items 3, 6, and 5 will appear together first on the display screen. Then,TEST1 containing items 7 and 5 will appear together, followed by TEST2 containing item 4 alone. You can switch from one display screen to the next in one frame, i.e. 16.6 milliseconds when running at a 60 Hz refresh rate.

The order of the test screens and the order of the items within the test screens are important, if the timing file contains commands to display multiple test screens simultaneously. TEST0 will always be on top, followed by TEST1, then TEST2, etc. (That is, TEST0 will cover up everything below it.) Within the tests, the items are drawn in the order they are listed. That is, if TEST0 lists items 1,2,3,4 (in that order), they will be drawn in the same order (1,2,3,4). Item 1 will be on the bottom, and item 4 will be on the top.

BCKGND. For the background, you list an item that you would like to have as the background on each test screen. Only the color of the item, as specified in the RGB columns in the items file, will be used.

TIMING. In this column, you list which timing file (state system file), you want to run with this condition. The timing files are imported separately, and are referred to by their number.

TRIAL_TYPE. Currently not used, except as the expected_response field in the output data file header.

FIX_ID. In this column, you list which item you want for the fixation stimulus. This is typically item -3, the item number that Cortex uses internally to hold the fixation stimulus. However, you may list any item.

---COLOR-PALETTE---. The color lookup table for images and movies can either be entered from within one of Cortex's menus, or listed on a line in the conditions file. This should be the name of a disk file that holds the lookup table. Each condition can have its own lookup table. Lookup tables can also be loaded by number from the state system, for dynamic changes in color during a trial.

We have neglected the issue of timing -- when and for how long the stimuli remain on the screen. Each experimental condition is associated with a timing file. The same timing file may be used by all conditions or each condition may have its own. The timing file is discussed in the next section.

4.4 Timing files

The timing file is responsible not only for informing Cortex when stimuli should go on and off, but also for any behavioral contingencies such as monitoring for a lever press, checking eye position, giving rewards, etc. The price for this flexibility is that the user must program every aspect of the flow of control, every contingency, every error state, etc.. If you know the C programming language, the state language of CORTEX will seem very familiar; the language is essentially a subset of C, with a few added features.

Each timing file is written as an ASCII file and saved with a file name. After you have written and stored your state systems as disk files, you then import one or more of the files into CORTEX, which sequentially assigns them each a number. These numbers are used as names for the timing structures. In the conditions file, you specify which timing structure is associated with each condition. For example, you may want to use two different timing structures in an experiment. Each will have its own ASCII file, with its own file name, and you will import (from within Cortex) the one timing file as timing structure #1 and the other as timing structure #2. In the conditions file, some conditions may list timing file #1 as the necessary timing structure and other conditions may list timing file #2. For information on writing the timing file, please refer to the separate document entitled, "Timing File Reference Manual".

4.5 Using the ITEMS, CONDITIONS, and TIMING files in a study

By modifying different ITEM, CONDITION, and TIMING files, either alone or in combination, you can create many different experiments. For example, once you have set up an experiment using a particular set of ITEM, CONDITION, and TIMING files, you could change the stimuli alone by importing a different ITEMS file into Cortex. Another way to achieve the same end would be to import a single large ITEM file that held a large number of stimuli to use, and then switch the items to use for a particular experiment by importing a new CONDITIONS file. The new CONDITIONS file will select new items from the ITEMS File. Alternatively, your primary interest might be in different experimental paradigms rather than different stimuli. In this case, you could keep the ITEMS and CONDITIONS files constant, but import different timing files for different experiments.

4.5.1 Deciding on your Experimental Design

Cortex can interleave trials from a large number of experimental conditions within a single experimental session. Each condition determines what happens on a particular type of trial. For example, in one experimental condition, you may want just a red item presented, and in a different condition you may want just a green item presented. Therefore, you would set up two conditions, each with a single item on the TEST0 screen.

Why use two conditions rather than one in this example? Why not just present the red item on the TEST0 screen and the green item on the TEST1 screen of a single experimental condition? You could do this, but then the order of stimulus presentations would be the same throughout the experiment (i.e. red followed by green). If you want stimulus presentations randomly interleaved, you should assign stimuli to different experimental conditions because Cortex can only randomize the order of presentation of conditions, not the screens within a condition (actually, strictly speaking, ANYTHING can be done within the state system, including randomizing

the order of presentation of screens within a condition, but this is tricky stuff). Another reason to use multiple conditions rather than a single condition with multiple screens is that there is a limit to how many screens can be presented sequentially within a single condition. You can only have 10 different screens of images (TEST0 through TEST9) within a single condition, but you can have any number of conditions.

O.K., you might ask, why do you ever need multiple image screens within a single condition? Why don't you just use one condition per screen image? One reason is that screens can be switched in 16 milliseconds (or less, depending on the refresh rate capabilities of the graphics card and monitor), whereas switching between conditions typically may require about a half a second because of various disk operations that must be performed. For some experiments, you may want to switch between certain stimuli rapidly, in which case you should put the stimuli on different screens within the same condition. Another reason is that you may want pairs or groups of stimuli always presented in the same order. Thus, on one condition you may want to have a red bar always followed by a green bar, and on a second condition you may want to have a blue bar followed by a yellow bar. Finally, if there are any behavioral contingencies linked to your conditions, at present these contingencies cannot easily span different conditions (except in some limited circumstances, described later). Consequently, if, for example, you expect a specific behavioral response to a red bar only when it has been preceded by a green bar, these two stimuli should be placed in the appropriate order within the same experimental condition. If you expect a different behavioral response to a red bar preceded by a red bar, you should place these stimuli in the appropriate order in a different condition. Items, incidentally, can be reused in many different conditions.

The concept of conditions is also related to how the data are stored. Recall that data are stored by trial and condition, with the data for each trial preceded by a header. Thus, one trial's worth of data in the data file would contain the data for condition one, the next trial's worth of data might contain the data for condition five (depending on the order of presentation), etc.. Within each trial's worth of data, the onset and offset of TEST0, TEST1, etc can be marked by using the encode() function in your timing file. Suppose that a red bar item is presented only on the TEST0 screen of condition 12. To locate the neuronal responses to the red bar in the data file, you would find all the trial data for condition 12 (the condition number is given in the header for each trial), and then find the time of occurrence of TEST0 in each of the trials for that condition. The spikes that fall within TEST0 onset and TEST0 offset represent the neuronal response to the bar on that trial.

4.6 Customizing Cortex with the Cortex.cfg configuration file

Within your current directory, you should also have a Cortex configuration file, which must be named 'Cortex.cfg'. This file contains information about the type and address of the hardware devices you are using as well as information about the size and location of graphic windows on the user's display (e.g. the histograms, rasters, eye position display, etc). The file is an ASCII file that can be edited both to conform to your current hardware configuration and to customize your display. An example configuration file should have been included with the distribution code.

The MONITOR_TYPE line of the Cortex.cfg file refers to the user's monitor. If this is not set correctly, your screen may go haywire when the program loads. For the SGT Pepper version of Cortex, there are fewer compatibility problems with the pepper card if you use the Hercules card as the user's display device. However, VGA should work, as well. For the dual computer version of Cortex, VGA should work fine.

A second critical piece of information to check is the name and address of the analog/digital I/O board in your system. This is set on the DEVICE line of the Cortex.cfg file, but the device information also is important on the THREADS and MULTI_SPIKE lines.

A third critical piece of information includes the graphics resolution and the pixels/degree that are to be used for the subject's display. These values are set in the GRAPHICS_SPECS line of the Cortex.cfg file. Because you specify all locations to Cortex in terms of degrees of visual angle, Cortex needs to know how many pixels equal one degree. You can calculate this based on the size, resolution, and distance of your monitor from the subject. Cortex assumes that your monitor has the correct aspect ratio, so that pixels are square. An example Cortex.cfg file can be found in Appendix C.

5. Running Cortex

5.1 The Cortex Menu Hierarchy

RUN	PLAY	ITEM	CONDITION	TIMING	LUT	DISK	SET
Parameters	Alone	Import	Import	Add	Number	Save	Import
Start	Condition	Display	View	Modify	View	Get	Load
Resume	Conds&save Data & hists	View/ Modify	Display	Delete	Activate		iComp
Display	Num play items	Use All	Modify	Help	Get		
ExternVars		Add	Save	Add Binary	Save		
Getaccuracy		Save		Modify Binary	Store		
Histogram		Reference			Recall		
Rasters							
Eog							
List epp							
Touch							
Eye trail							
Clear							

5.2 Main menu

After Cortex loads, the following menu will be displayed:

Run Play Item Condition Timing LUT Disk Set

To choose a command from the menu, either type its highlighted character or highlight the entire word with the arrow keys, followed by the "enter" key. The "escape" key will bring you out of the current menu and back to a higher level. Normally, you will want to import the items, conditions, and timing files first.

NOTE: From any menu you can hit the F1 key, which will put you into DOS while retaining Cortex in memory. From DOS, you can run a DOS editor to change any items, conditions, or timing files you want, or you might execute a DOS command such as a directory listing, or whatever. To return to the program, just type 'exit'. Everything you have entered into the program is saved when you return, as is any data you have collected. If you changed an items, conditions, or timing file while in DOS, Cortex will not know about your changes unless you re-import the file into Cortex.

Item. To import the items file, type 'i' for items, and Cortex will display the items menu. Then choose 'i' for import, and give the file name. To actually see one of the items on the graphics screen, type 'd' for Display, and give the item

number when prompted. If the items file contains images, you will need to import a color lookup table to see the image. Type 'l' to invoke the LUT menu, then give the filename for the table. If you simply want to view a listing of the item's properties (item type, length, width, etc.), or change them, type 'v' for view/modify and give the item number when prompted.

Another items menu choice is the reference point. If you type 'r' for reference, Cortex will tell you what is the current reference point and will then ask if you want to change it. The five possible reference points are: 'background (-4)', 'fixspot (-3)', 'Rfield (-2)', 'Rpoint (-1)', and 'Play (0)'. The reference point is the location that Cortex defines to be the center of its coordinate system for placing items on the screen. For some studies in fixating subjects, the fixation spot is the most natural reference point. If you make the fixation point the reference point, then the location of all the stimulus items will necessarily be in retinal coordinates. Alternatively, you might want your items displayed at locations relative to some other point on the screen, such as the center of the receptive field you are studying. If so, choose 'Rfield' as the reference (ie. Item number=-2). Note that, unlike other items, the fixation point location is always specified in screen coordinates, i.e. its location is not referred to any reference point other than itself. The reference point and its location can also be changed within the "play" routine, which is described a little later.

Condition. From the main menu you can access the Conditions menu just like you did the Items menu. Like with items, you can ask Cortex to display a condition. One difference from the items display is that when you ask Cortex to display the items listed on a particular test screen of a particular condition, it will show you all of the items that you linked together on the test screen, including the background item. Thus, you can see how the stimulus screens will actually appear during a trial.

Timing. Like the Items and Conditions, you access the timing menu from the main menu. Unlike the case for Items and Conditions, you can import multiple timing files. To import the first timing structure, type 'a' to add a timing file, and you will be prompted to supply the filename of the file. After importing the first file, type 'a' for Add, to add more timing files to the list. If you make a mistake, you can go back and reenter a state file, by typing 'm' to invoke the modify option. Each condition in the Conditions file must specify the number of a valid timing structure.

Disk. Rather than importing the same files each time you run an experiment, you can save all of the files and other parameters in a single disk save file, which you can import the next time you run the program. Go to the disk menu and type 's' for Save. Normally, you would not 'save' the files and parameters until you have entered all of the information required by Cortex, which is described below. To retrieve the file at a later time, type 'g' for Get in the Disk menu.

Play. When you are collecting data, Cortex runs in a very automated fashion and does not allow you to change stimuli interactively. However, it is often desirable in mapping receptive fields to be able to change stimuli and their location "on the fly". In such an informal mode, you generally don't need to be collecting and storing data, since you are changing stimuli in an unpredictable way. Rather, you want to be able to hear the neuronal responses on a loudspeaker while you are playing around with stimuli. After you have decided on the receptive field, you want to be able to use the receptive field center as the reference point for all subsequent stimulus presentations when you collect data. You may even want the play stimulus you have created to become an item that is displayed automatically when data are collected again. All of this is what PLAY is designed for. Play has two modes, "alone" and with "conditions". The "alone" mode is appropriate for experiments in which there are no behavioral contingencies. By default, Cortex uses a white square as the Play item. In this mode, PLAY simply allows you to move and change stimuli constantly, using a mouse.

After the Play mode has started, the F10 function key will invoke a full-screen list of play parameters. The arrow key on the keyboard can be used to move the cursor through the different parameters. Type in new values to change them. Some of the parameters (Position, Item Size, and Orientation) can also have their values controlled by the mouse. For example, if "Position" is highlighted, pressing and holding the right mouse button, while moving the mouse, should move the Play item left and right. Similarly, pressing and holding the left mouse button, while moving the mouse, should move the Play item up and down. Pressing and holding both mouse buttons, while moving the mouse, should allow the Play item to move in all directions.

These same operations will work if the cursor is on the "Item Size" choice, except they will change the size of the item. "Orientation" also can be manipulated by the mouse, but the left and right mouse buttons both have the same effect on orientation. Once the cursor is on "Position", "Item Size", or "Orientation", you can hit the F10 key again to make the full-screen parameter list disappear. Whatever parameter was highlighted will be the parameter that the mouse will change. If the cursor is not on one of these three parameters, the mouse will have no effect.

In the "conditions" mode, PLAY runs whatever experimental conditions you choose, and simultaneously allows you to change and move stimuli around with the mouse. The experimental conditions in this case normally have some behavioral component to them -- e.g. a fixation task. In this mode, PLAY will continue to reward the subject for performing the task correctly and will operate just as it does when you are collecting data. The difference is that you have control over your own mouse-driven stimulus.

When you have found a particular receptive field location that you want to save, PLAY allows you to save the play item parameters, including its location, to any item in your item file. The item you save the play item to, can also be the reference item. Reference point items are described in the section on the Item menu. From within PLAY, you can select 'receptive field' or another item to be the reference point, and then save the location of the mapping stimulus to the reference point. Then, when you return to collecting data, the items will be dis played at locations relative to the center of the receptive field that you found. The reference point can also be changed from within the ITEMS menu, described previously. NOTE: This code has not been changed since the last release, but during recent testing of multiple versions we have not been able to get the reference item and reference point to function in the manner described above.

LUT. The LUT menu is used for loading, viewing, and changing color lookup tables in Cortex. Refer to Section 7.3 for more information.

5.3 The Run Menu

Run. After importing the necessary files, you go from the main menu to the menu for running trials and collecting data. The run menu is the following:

Parameters Start Resume Display ExternVars Get accuracY Histogram rAsters eOg List epp touch eye_Trail clear

Parameters. The parameters menu allows you to set up staircasing, and to set other general parameters.

Some of the more obscure options are as follows:

Blocking. Cortex allows the conditions to be grouped within blocks. Within each block, you can specify how many trials you want to run. This has great utility when you want conditions to be randomly chosen within a small subset of the conditions, but not across the full range of conditions. For example, you might set up one set of conditions in which the subject is to perform visual discriminations based on color and another set of conditions in which the subject is to perform discriminations based on orientation. By placing the color conditions within one block and the orientation conditions within another, you can prevent the orientation conditions from being interleaved with the color ones.

Selection with/without replacement. If you choose 'without replacement', then when Cortex picks a condition to run, it deletes that condition from the list of conditions that it will choose from for the next trial. Cortex will calculate how many times each one can be called for the maximum number of trials. Then, the conditions can be presented in any order. Each condition has a counter of how many times it can be presented. The counter is changed each time the condition is presented. So, ultimately (across all trials) there is an even distribution. If you choose ' with replacement', then Cortex puts the chosen condition back into the available list of conditions, so that condition might even be chosen again for the next trial.

On Error. In addition to selection with/without replacement, there are other blocking options that affect how conditions are chosen. Some of the possible condition-picking options are ignore, retry_immediately, and

delayed_retry. The last option is extremely useful, but it requires some explanation. When training a subject, it is often desirable to repeat a condition that it gets wrong, so that it doesn't decide to just skip all the difficult conditions or make just one kind of response. A common technique is to use "correction trials", which means to simply repeat a condition that is not performed correctly. The problem with conventional correction trials is that a subject who makes an error can learn to simply make the opposite response on the next trial, and it will then always get the next trial correct. A better way to do correction trials is to put the condition performed incorrectly back into the list of conditions to be chosen from randomly on the next trial. This option assumes random selection without replacement for all correctly performed trials. The incorrectly performed condition will keep coming up in the block ever more frequently (but not necessarily the very next trial) until the subject gets it correct. (More information on Blocking can be found in Appendix A.)

Eye Window Size. For studies in fixating subjects, you need to specify the size of the fixation "window", or fixation criteria. If, in the timing file you wrote that the subject must start to fixate by some time point during the trial, then Cortex will use the window size to decide whether fixation has been achieved (and remains achieved). To open the window all the way up, give it a size of 30 degrees or more.

A2D Gain and Offset. Cortex cannot actually measure degrees of eye position. It can only measure voltages from the A/D converters. Cortex assumes that the full-scale range of the A/D converter is equivalent to an eye movement from one side of the display screen to the other. If this is not so, you should adjust the eye position gain either using the gain setting on your eye tracking hardware, the gain setting on the analog input board, or let Cortex set the gain programmatically using the value supplied in the Cortex.cfg file as the A2D_GAIN parameter. Refer to the documentation for your particular board for information on how to set this parameter. Typically, you shift your fixation stimuli 5 degrees or so to the left, right, up, and down, and keep adjusting the eye coil hardware gain and offset setting until Cortex shows that the subject's eye position is on it. The size, in degrees, of the "crosshair" on the eye display can be changed by setting the "Eye max saccade allowed" parameter in the Run:Parameters:General menu, which is a convenient way to measure distance in degrees on the display screen. If for some reason you can't get your gain high enough or whatever, you can also enter an integer gain multiplication value in the Run:Parameters:General menu, which will be used to multiply all incoming A/D values. The default is 1, which is the recommended setting. There are also eog_xoffset and eog_yoffset parameters in the Run:Parameters:General menu, which will be subtracted from the data. The default for the offset parameters is 0.

Start. This menu option starts running the experiment. The items, conditions, and timing files should have already been loaded. You will be prompted for the name of a file to use for the output data file.

Display. Displays the encodes of the trial that just completed.

Extern Vars. This menu option is used to load, set, and modify certain Cortex State System external variables. There is more information about using external variables in the "Timing File Reference Manual".

Get. Retrieves saved parameters, items, conditions, and timing files. (It has the same behavior as the Disk:Get menu option.)

EOG. Lists the EOG values from the trial that just completed, if the parameters and timing file were set to collect EOG data.

List epp. Lists the EPP values from the trial that just completed, if the parameters and timing file were set to collect EPP data.

Touch. Lists the touch screen values from the trial that just completed, if the parameters and timing file were set to collect touch screen data.

Eye Trail. Connects the dots of the EOG values on the user's monitor, if the parameters and timing file were set to collect EOG data.

Accuracy. Lists the accuracy and circular buffers for the blocks and conditions.

Clear. Truncates the data file, clears the histograms and raster plots, and clears the staircasing data.

5.4 Quick Start Instructions

If the Cortex executables have been downloaded and installed on your computers (see Section 3), these instructions will run you through a brief introduction to the use of Cortex. These instructions assume that you have installed Cortex in a directory named \cortex.

5.4.1 Dual computer version (with DirectX-based receive program)

Make sure that the send computer has been started up in true DOS mode (Command prompt only).

1) On the receive computer:
   a) DXRECV.EXE is a Windows console application, which must be run from Windows (not DOS). The easiest way to run it, is to make a shortcut on the desktop of the receive computer to the executable, \WCORTEX\DXRECV.EXE. When you want to run the program, just double-click on the DXRECV.EXE shortcut icon. An hourglass may (or may not) appear for a few seconds. When you start the send computer, the receive program should go black.
2) On the send computer:
   a) At the DOS prompt, type "cd \cortex\remote32" and hit enter.
   b) Then, type "wcsend" to start the Cortex send program.
   c) Cortex will ask if the parameters are OK. Type "Y" for yes.
   d) A connection should now occur between the send and receive computers. It will now ask if you want to run Cortex. Type "O" for OK.
   e) Now, you must load an items file, a conditions file, and a timing file. You can find examples of these files in the c:\cortex\demos\ directory. To load an example items file, choose Items:Import from the menu, and then type in c:\cortex\demos\misc\gtest.itm and hit enter. Hit "escape" until you get back to the top level Cortex menu. To load the conditions file, choose Conditions:Import from the menu, and then type in c:\cortex\demos\misc\gtest.cnd and hit enter. Again, hit "escape" to get back to the top level. To load the timing file, choose Timing:Add from the menu, and then type in c:\cortex\demos\misc\gtest.1 and hit enter. Again, hit "escape" to get back to the top level.
   f) There are a bunch of parameters that may be set. However, to actually run an experiment, you only absolutely must tell Cortex how many times to run, and with what conditions. To do this, choose Run:Parameters:Block/repeat:Individual blocks. A box will appear with more choices. For now, just choose the following:
      1. Condition Order: Incremental
      2. On error? : Ignore
      3. Max trials:4
      4. First cond: 1
      5. Last cond: 4
   g) Hit enter after each of these choices. Then, hit escape to go back to the top level of Cortex.
   h) Now, run the trial by choosing Run:Start. It will ask you for a data file prefix. You can type any 8-character name here. The first time that you run with the given data file prefix, Cortex will append a ".1" to this name, and ask you if it is OK. Choose OK and the program should start. If you start again with the same data file prefix, this file suffix will be incremented by 1 each time.
   i) For this gtest.1 timing file, it is coded so that you must hit the space bar to start the item moving around on the screen. Hit the space bar again to switch to the next item. (Note that since we only told Cortex to run 4 trials above with conditions 1-4, it will only change four times.) When it is finished, as usual, hit "escape" to get back to the Cortex menus.
   j) To quit Cortex, hit Escape until it says "Exit Cortex? Yes No". Choose Yes.

5.4.2 Dual computer version (with Scitech-based receive program)

Make sure that both the send and receive computers have been started up in true DOS mode (Command prompt only).

1) On the receive computer :
   a) At the DOS prompt, type "cd \cortex\remote32"
   b) Then, type "rvesactx" to start the Cortex receive program.
2) On the send computer:
   a) Follow the instructions for the send program in section 5.4.1.

5.4.3 NNIOS version

Make sure that the computer has been started up in true DOS mode (Command Prompt only). At the DOS prompt, type "cd \cortex\nnios". Then, type "nncortex" to start the Cortex program. Follow the instructions in section 5.4.1, from step 2), part c) onward.

5.4.4 TIGA version

Make sure that the computer has been started up in true DOS mode (Command Prompt only). At the DOS prompt, type "cd \cortex\num9gxi". Then, type "ticortex" to start the Cortex program. Follow the instructions in section 5.4.1, from step 2), part c) onward.

5.4.5 SGL32 version

Make sure that the computer has been started up in true DOS mode (Command Prompt only). At the DOS prompt, type "cd \cortex\remote32". Then, type "sgl32" to start the Cortex program. Follow the instructions in section 5.4.1, from step 2), part c) onward.

6. Limitations of Cortex due to structure and hardware

6.1 Data storage

Although the details of the program will be discussed later, one important detail relevant to the capabilities of Cortex is the manner in which data are stored. Cortex currently keeps all of the raw data (time of occurrence of each spike etc.) for an experimental run in a disk file rather than in memory, except for a one-trial buffer. While this allows both for a virtually unlimited amount of data to be collected and for a simple data-saving module, it would be too time-consuming to sort through the disk file to resynchronize the histograms and rasters to a new time or event on-line during an experiment. Therefore, the synchronization of the on-line rasters and histograms is set and controlled by the state system and cannot be resynchronized during an experiment. You can, of course, do anything you want with the raw data outside of Cortex itself, after the data have been collected.

6.2 Data file format

Each trial's worth of data is preceded in the disk file by a "header" that contains information such as the experimental condition number, the type of behavioral response that was expected of the subject (if any), the type of response the subject actually made, whether the trial was performed correctly, the types of errors that were made, and the sizes of various arrays that follow. One array contains the codes for all events that occurred during the trial, including spikes, stimulus presentations, behavioral responses, etc. A second array contains the times of occurrence of each of the corresponding events. Time is measured from the beginning of the trial, and is represented by a 32 bit long integer. Additional arrays may contain eye movement or evoked potential analog data. The information in the header can be used to quickly sort trials for later data analysis. For example, suppose you want to know the individual firing rates to each of the stimuli presented on each experimental condition, for correctly performed trials only. This can easily be done using a combination of information in the header together with the information in the data arrays.

It is important to note that histograms are not stored in the disk data file. Averaged histograms can be recompiled off-line from the data in the arrays, using other Cortex data analysis programs.

6.3 Interrupt structure

Cortex currently runs with a single interrupt routine, which both updates a software clock and samples all data lines. The interrupt is initiated by the programmable system clock on the PC. This clock normally interrupts every 55 milliseconds when MS-DOS is in control, but we re-program the clock to interrupt every 1 millisecond (optionally 0.1 millisecond), and we set the interrupt vector to point to our own service routine. We don't allow the MS-DOS clock interrupt service routine to operate when Cortex is in control; thus, time stands still as far as MS-DOS is concerned. The reason we do this is that the PC has a primitive interrupt handling system compared to the PDP-11, and we think it is unwise to allow any interrupt routine to gain control of the CPU except for Cortex's own critical clock service routine. The control of behavioral contingencies and the presentation of stimuli all take place outside of the service routine.

6.4 Graphics display hardware (Sgt. Pepper)

While the Pepper SGT has been a workhorse graphics card for neurophysiology, Number Nine Computer Company decided in November, 1993 to no longer manufacturer it. The following is a description of the SGT's capabilities, which must be emulated in any future replacement.

The Pepper SGT has some very nice features for visual neurophysiology. For one, it operates independently of the graphics card that puts up the user's display, so that the user and experimental subject can view different screens simultaneously. Secondly, it can simultaneously display 8 bits of color or gray scale from a 24 bit palette, which is marginally adequate for stimuli such as sine-wave gratings that must be "linearized" in intensity to correct for CRT displays, which are always highly non-linear. Third, you can have up to 4 megabytes of memory on the card

(although 1 MB is sufficient for most purposes), which gives you plenty of space to store patterns and also room to "roam" over patterns. Fourth, it runs at 60 Hz non-interlace, with 640 x 480 resolution, which, again, is marginally adequate for neurophysiological work in most cortical areas (the 60 Hz might be a problem in the LGN). Fifth, the Pepper SGT informs the program of when it begins a vertical sweep of the screen, so that stimulus presentations can be timed to 1 millisecond accuracy (That is, you can know when a stimulus is presented to 1 millisecond accuracy. You have less control than this over the presentation time, because the SGT might be in the middle of a sweep at the precise time you want your stimulus to come on. Stimuli can be switched on only at the start of the sweep.) Finally, it has the Intel 82786 graphics chip which has hardware support for windows that have independent sizes and positions. Thus, for example, you can keep one stimulus stationary on the screen (such as a fixation stimulus) in one window while other stimuli move around in different windows (analogous to sprites), or you can scroll a grating inside one window in one direction while scrolling another grating in another window in a different direction. A major decision for us was to put the stimulus display card in the same machine that collects the data. The alternative (now implemented as the "two-computer" version of Cortex) is to use two PCs, one for data and one for graphics, with communication between them.

Because it is often necessary in visual neurophysiology to switch rapidly between two or more different images, each of which is large and complex, we decided that the stimuli to be presented on a given "trial" in an experiment would be drawn in different portions of the graphics memory before the trial begins. This allows us to change to a new image within one video frame (16 milliseconds, at a 60 Hz refresh rate). Note, that we do not store all of the images for an experimental_session in graphics memory, because there is no limit on the number or size of stimuli in a session, and, thus, we could not always be sure that they would all fit in. Rather, we just load in the images for a single trial at a time. Optionally, you can instruct Cortex (within the state system) not to unload an image at the end of a trial.

7. Image File Formats and Color Lookup Tables

7.1 Image File Formats

7.1.1 The Cortex Image File Format

The Cortex image file format is a binary file, arranged as successive rows of 8 bit pixel values, (i.e. the first horizontal row of pixel values, followed by the second row, etc). Preceding the image data is a header describing the image file. Byte location 10-11 should contain the pixel depth of the image. At byte location 12-13 in the file you should put the X size (i.e., the number of pixels in width), in bytes. At location 14-15 should be the Y size (i.e., the number of pixels in height). In addition, at byte location 16-17 should be the number of frames, for movie items. Subsequent frames of the movie, or standalone image files should just contain a zero in byte 16. The image data itself should begin at byte location 18 in the file.

This information is summarized as follows:

Byte Locations in File

0 - 9	10 - 11	12 - 13	14 -15	16 - 17	18 to end
blank	Pixel Depth	X size	Y size	number of frames	Image data

To convert the other image format files into Cortex image format files, there are a number of utilities that are provided "as-is".

If the files are stored as TIFF or JPEG: On the Cortex web site (from the Documentation page) there is a link to the Matlab routines. You will need to traverse the directory structure until you get to \pub\cortex\matlab\bj-img-func\. There is a document named "mat-img-functions.doc" that tells how to use these functions to go from TIFF or JPEG into Cortex image format. These routines can change a whole directory of TIFF or JPEG files to CTX files at the same time, and create a single LUT file. This is important, since only one LUT is allowed per condition in Cortex. In the graphics program that is used to create the images, the images should be saved in RGB Color and 8 bits per channel. (In Adobe PhotoShop, these settings can be made in the Image:Mode menu.)

If the files are stored as BMP, HDF, JPEG, PCX, TIFF, or XWD: Follow the links on the Cortex ftp site to \pub\cortex\matlab\salk\saveload. There is a document named "vclimg.doc" which describes how to use these routines. These routines can be used to convert Cortex image files from other formats, or into other formats from Cortex images.

If you do not have Matlab, the only other program is one for use with TARGA (TGA) files. There is a DOS program called TARGCORT.EXE up on the Cortex ftp site in the \pub\ cortex\targcort directory which will convert the images from TARGA ->Cortex. (Of course, you can convert your BMP or other image format files to TGA using a commercial graphics program, and then use targcort.exe.)

7.1.2 Image File Formats Available with Scitech(MGL) version of Cortex

Scitech's MGL library can load BMP, JPEG, and PCX files, in addition to the standard Cortex (CTX) image files. The type of bitmap file is determined by the file extension. That is, to be processed correctly: JPEG files must have the extension .JPG; BMP files must have the extension .BMP; and PCX files must have the extension .PCX. All other bitmap file extensions are assumed to be the standard Cortex (CTX) bitmap type. Therefore, this naming convention should be transparent if you continue to just use CTX type bitmaps. It is assumed that the CTX files follow the documented format from section 7.1.1.

If you want to change the LUT for an image, it is almost mandatory to use CTX files. The other types of images (i.e., .BMP, .JPG, and .PCX) keep their LUT internal to the image file. For non-.CTX file cases, you would need to know the internal structure of the image file and where the individual pixels were located. Even then, it may not work to change the Cortex LUT, and expect the bitmap color to change.

7.1.3 Image File Formats Available with DirectX version of Cortex

The DirectX receive program can currently only load BMP files, in addition to the standard Cortex (CTX) image files. Like the Scitech version of Cortex, the type of the bitmap file is determined by the file extension. That is, to be processed correctly, BMP files must have the extension .BMP. All other bitmap file extensions are assumed to be the standard Cortex (CTX) bitmap type. Therefore, this naming convention should be transparent if you continue to just use CTX type bitmaps. It is assumed that the CTX files follow the documented format from section 7.1.1.

In the DirectX receive program, .bmp files in 8 bpp will use the Cortex LUT. (For example, if the LUT is not in the conditions file, then the image will be displayed in black and white.) It is therefore recommended to create a default LUT that contains the Windows system palette, for example, for use in 8 bpp. At greater than 8 bpp, the images will look fine, and will not need/use the Cortex LUT. There is a utility program, bmp2lut.exe, which extracts the LUT from a .bmp file. It is available "as-is" from the Cortex ftp site.

7.1.4 Transparency with DirectX receive program

For the DirectX receive program, transparency works with both .BMP and CTX image files. The background color as specified in the items file is used as the "source key" color. All pixels in the image that are this color (i.e., LUT entry #0), will be transparent when displayed in Cortex. The files \cortex\demos\dx\dxtransp.* show this effect nicely.

7.2 Movie File Formats

7.2.1 Cortex Movie File Format

For animation, such as dynamic motion displays, you may concatenate any number of (CTX) images in the same disk file, and you can instruct Cortex to display them sequentially (or any order you like). Each frame should be organized like the Cortex image file format described above. That is, each frame should have a header, which sets the x and y size, the pixel depth, etc.. The number of frames in the movie only needs to be set in the first frame's header. For the two computer versions of Cortex, the number of frames field must be (number of frames minus 1, since it counts the frames as starting from zero.)

7.2.2 Movie File Formats Available with DirectX version of Cortex

The DirectX version of the "receive" program (in two computer Cortex) has the ability to play AVI, MPEG, and QuickTime movies, in addition to the standard Cortex (CTX) image file format based movies. The type of movie file is determined by the file extension. Cortex gets the filename and item TYPE ("11" for movies) from the items file. To be processed correctly: AVI files must have the extension .AVI; MPEG files must have the extension .MPG; and QuickTime files must have the extension .QT or .MOV. All other movie file extensions are assumed to be the standard Cortex (CTX) movie type. Therefore, this naming convention should be transparent if you continue to just use CTX type movies.

When you display a movie in Cortex, it reads all of the frames of the movie into memory. Then, if you want to loop through the movie for a specific period of time, it just uses the memory copy to play over and over again. AVI, MPEG, and QuickTime movies use DirectShow to uncompress/read the movie frames. It takes a long time to load an AVI file (at least 20 seconds to load a 75-frame movie). Therefore, it is definitely better to make the movie as short as possible (i.e. 2 seconds) and then tell Cortex to loop for 10 seconds, if the movie actually contains a repeating cycle inside the movie file.

Since the movies are very slow to load, you might want to use the "dont_unload_conds()" function in your timing file, if the movie is to be used in each trial.

Since it uses the DirectShow component of DirectX to display the movies, Microsoft Internet Explorer version 4.0 or higher must be installed on the receive computer.

Not all video codecs are supported by DirectShow. The list of video media types that are supported with DirectShow can be found at the web site: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directshow/htm/thedirectshowsdkandthewindowsmediaformatsdk.asp.

Colors for the (non-CTX) movies look best when you are running at 32 bpp, but it makes the movie slower to load. For 8 bpp, you will need to generate a LUT that will look good with the movie.

Only the video track of the movie file will be played by Cortex. The audio track will be ignored.

7.3 Color Lookup Table

7.3.1 What is a Color Lookup Table?

In order to display an image on the screen, the program must tell the graphics hardware how to color each pixel that makes up the image. For an "8-bit image", for example, each pixel has a color which is described by 8-bits of information. 8-bits of information can specify 256 different values. As you know, a computer can display millions of different colors (i.e. more than 256!). To allow as much customization as possible for the 256 colors, color lookup tables were invented. All a LUT really is, is a matrix of red, green, blue values that tell which colors you want to make up your image. For instance, a LUT file might look like this:

INDEX	RED	GREEN	BLUE	(resultant color)
0	255	255	255	white
1	0	0	0	black
2	255	0	0	red
3	0	255	0	green
4	0	0	255	blue
5	160	0	160	light purple
6	100	100	100	gray
etc.				

Your image file now uses its 8-bits/pixel to give an index into this matrix. That is, when the pixel should be black, the image file will contain a "1". If it should be green, it will contain a "3", etc. You can probably see how a LUT would be useful if you wanted to change the colors of your image. Let's say you wanted to change all the black in your image into green. Instead of changing anything in your image, you could simply change the LUT so that index[1] contained (0,255,0) instead of (0,0,0).

7.3.2 How Cortex uses Color Lookup Tables

Cortex uses the first 128 of the 256 rows in the lookup table for single-color items such as bars and circles, and the upper 128 rows to hold the color values for bitmapped patterns stored in disk files. Each of the single-color items in a given condition gets one row in the lookup table. The user enters the red, green, and blue values for each of these items in the items file. (Note: The actual row assigned to the item in the lookup table is irrelevant to the user, and therefore is assigned by Cortex using a specific algorithm. To retrieve this information, call the function ITEM_POSlut_index() in your timing file.)

For the bitmapped items, which are necessarily 7 bits "deep", you must fill in the upper 128 rows of the lookup table. This table of values can be created off-line, or internally through the LUT menu. Cortex must be given the

name of the file that holds the values, either in the conditions file, or by loading it via the LUT menu. The file is organized as a series of four successive integers for each of the 128 table locations that will be used. The integers hold the red, green, and blue gun intensities, plus a dummy integer. The user has the option of specifying one "experiment-wide" lookup table that will be applied to all bitmapped items, or of assigning a unique lookup table to each experimental condition. At the appropriate time, the lookup table values are read by Cortex into the graphics card memory. These lookup tables can be full-color tables or gray-scale tables, depending on the needs of the user.

For achromatic sinewave gratings, the user will normally "linearize" the output on the display CRT by adjusting the lookup table values so that equal increments in pixel values will produce equal increments of intensity on the CRT screen. It should be noted that in the current version of the program, if the user wants two or more bitmapped full-color items on the screen at the same time, they must use the same lookup table.

7.3.3 Using the Cortex LUT Menus

In the conditions file, there is a column for the "Color Palette". Each condition can only have one color lookup table; so if you have multiple items in that condition, they will all use the same lookup table. Cortex can handle multiple LUTs in memory, but each LUT must be assigned a number. The number of LUTs that can be loaded into Cortex is dependent on the memory available in the computer. Cortex will try to allocate memory for the number of LUTs that you request.

As mentioned above, Cortex uses the bottom 128 values of the color lookup table to store the color of the background, and the colors of any of the "generic" items (such as, bars, circles, etc.) You can override this behavior by loading the LUT starting from index 0. To do this, you must first tell Cortex how many LUTs that you have (using the LUT:Number menu). Then, load the LUTs using LUT:Get:From Disk. It will display the values in the LUT. You can page through them, and then hit ESC to get out of the list. It will then prompt you for the LUT number, the starting index, and ask if you want to activate the LUT on the graphics card (choose "yes"). If you want the starting index of the LUT to be 0 (instead of the default 128), you must go into the Run:Parameters:General menu and set the "Load CLT from what index?" parameter to 0 instead of 128. (The load_CLT() function has the same effect of loading the CLT at index 0.)

To change a lookup table, once it is loaded, you can use the LUT:View menu. You can just go in and type the new values. When you are finished, hit ESC. The changes will just be held in memory. If you want them to be actually written to disk, you must choose LUT:Save.

Lookup tables are usually created by taking an existing table, and then changing/saving the values to a new file (as outlined in the previous paragraph). A lookup table can also be created with a binary editor (or a C or Matlab program) in which you would write each value of the table. The lookup table is just made up of short integers for the red, green, blue, and a dummy value (just make it zero), for each index of the LUT. (That is, index 0, followed by index 1, etc.) . The Cortex website has a Matlab routine that will write a Cortex LUT from a given matrix. (See http://www.cortex.salk.edu/CortexMatlab.php.)

8. Source Code

8.1 Distribution

The Cortex program includes over 35,000 lines of source code in about 200 source files. The DOS components of the program have been compiled with Microsoft C/C++ version 5.1, and Watcom C/C++ version 11.0. The Windows component (i.e., the DirectX receive program) has been compiled with Microsoft Visual C++, version 5.0. The source distribution can have any of a number of different formats.

Prior to version 5.7 of Cortex, the source distribution was in a compressed form and had to be "unarchived" by the "Arj" program (i.e., arj.exe), which can be found on the Cortex ftp site. Arj will create directories on the hard disk during the unarchiving process.(The user begins by creating a new directory. For Cortex V4.32, for example, the directory name might be "CORTEX4.32". Once the directory is created, the archive file (e.g. CORTEX.ARJ) and the archive program (e.g. ARJ.EXE) are placed in that directory. Then, the file is unarchived with the command: arj cortex in this example.)

Currently (i.e., version 5.7 and higher), code is compressed into the "Zip" format. Zip files can be uncompressed with commercially available products, such as WinZip and PKunzip. After the Cortex distribution zip file is unzipped into a directory, a listing of the directory should look something like this:

```
 Volume in drive C has no label
 Volume Serial Number is 07CE-0A09
 Directory of C:\c594

.              <DIR>        11-24-00 11:11a .
..             <DIR>        11-24-00 11:11a ..
CORTEX5  TXT         3,546  09-13-97 12:05p CORTEX5.TXT
DEMOS          <DIR>        11-24-00 11:11a DEMOS
DOCS           <DIR>        11-24-00 11:11a DOCS
DXSOURCE       <DIR>        11-24-00 11:11a DXsource
NUM9GXI        <DIR>        11-24-00 11:11a NUM9GXI
REMCORT        <DIR>        11-24-00 11:11a REMCORT
REMOTE32       <DIR>        11-24-00 11:11a REMOTE32
SGTPEPPR       <DIR>        11-24-00 11:11a SGTPEPPR
SOURCE         <DIR>        11-24-00 11:11a SOURCE
UTIL_BIN       <DIR>        11-24-00 11:11a UTIL_BIN
WCORTEX        <DIR>        11-24-00 11:11a WCortex
         2 file(s)          3,546 bytes
        12 dir(s)         429.53 MB free
```

8.2 Components

The following directories hold the source code, executables, and demos for Cortex:

\demos	files to demonstrate the functionality of Cortex
\docs	documentation files for Cortex
\dxsource	source code for the DirectX receive program
\num9gxi	executables for the #9GXi-TC graphics card version of Cortex
\remcort	executables for the 16-bit two-computer version of Cortex
\remote32	executables for the 32-bit two-computer version of Cortex
\sgtpeppr	executables for the Number Nine Sgt. Pepper graphics card version of Cortex

\source	source code for the DOS components of Cortex
\util_bin	executables for some utility programs for Cortex
\wcortex	executable for the DirectX receive program

8.3 Compiling the executables

1) To build the 16-bit version of Cortex (that is, NNCORTEX.EXE, TICORTEX.EXE, or CORTSEND.EXE), the compiler is Microsoft C/C++ version 1.52. The MAKEFILE is in the \cortex\SOURCE directory. At a DOS prompt, set up an environment variable called CORTDIR, by typing "SET CORTDIR=c:\cortex" (or wherever your Cortex directory is located). Then, type: SET INCLUDE=%CORTDIR%\SOURCE\INCLUDE;%CORTDIR%\SOURCE\LIB16MSC;C:\MSVC\INCLUDE to set the include path. Change to the \cortex\SOURCE directory, then type "nmake all > out.txt". When it is finished, look at the out.txt file and see that it ran successfully. (There will be warnings, but there should not be any errors.)
2) To build the 32-bit DOS version of Cortex (that is, WCSEND.EXE, RVESACTX.EXE, and SGL32.EXE from the \REMOTE32 directory), the compiler is Watcom C/C++ version 11.0. (Note: After you have installed the Watcom C/C++ compiler, you will need to apply the patches to the compiler. The patches are available from the Sybase web site.) Using the Watcom IDE, open the "send" project \cortex\source\w_csend\wcsend.wpj. Using the IDE menu, mark all targets for remake. Then, select "Make all". Repeat this for the "receive" project \cortex\source\w_vesa\rvesactx.wpj. Repeat this for the "single-computer 32-bit no-graphics" project \cortex\source\w_sgl32\sgl32.wpj.
3) To build the 32-bit Windows DirectX Receive program (DXRECV.EXE), the compiler is Microsoft Visual C/C++ version 5.0. You will also need to download and install the Microsoft DirectX SDK from the Microsoft web site (http://www.microsoft.com/windows/directx/default.aspx?url=/windows/directx/downloads).
    a) Open the Microsoft Visual C/C++ IDE. Then, File:Open Workspace:Dxrecv\DXrecv.dsw.
    b) Open the Tools:Options:Directories dialog panel. Check that the Include file and Library files have the proper paths to the code. Besides \source\include, the \dxsource\include path must also be given. The \dxsource\lib32msc directory contains the library for the Windows serial communications. Please note that the licensing of the DirectX SDK does not allow the redistribution of the DirectX libraries. Therefore, if you want to rebuild DXRECV.EXE, you must download the DirectX SDK and the DirectX Media SDK from the Microsoft web site.
    c) Open the Project:Settings:Post-build step dialog panel. Check that it copies the executable to the proper directory.
    d) Choose Build:Set Active Configuration: Release. Then, Build:Rebuild All. After the project has built successfully, close the project.
    e) If you would like to build a debuggable version of DXRECV.EXE, repeat the build step, but this time Set Active Configuration to Debug. Rebuild All again. Note that only the Release executable gets copied to the WCortex directory, unless you change the Project:Settings:Post-build step in the Debug build Project settings. The debug workspace configuration includes the settings necessary to generate an executable that can be debugged using the NuMega SoftIce debugger. The release workspace configuration does not include debug information, so it builds a smaller (and supposedly faster) executable.

8.4 Adding new functions and external variables to Cortex

While functions can be written into a state file, it is sometimes more advantageous to incorporate the functions into Cortex itself. For example, to implement the natural log function, it is easier to just use the built-in C log function than to write a complete state file function to calculate logs. Using the built-in C function will also run a lot faster.

8.4.1 Adding new system functions to Cortex

The first step is to create the new function itself. The function can be added to the file _userfns.c. It might be more useful to create a new file for all your new user functions, using _userfns.c as a guide. After entering all the appropriate #include, extern, and #define statements, a typical function entry is added like this:

```
float CSS_polar2x(CSS_ARGS) /* name always starts with CSS , arguments are always
CSS_ARGS */
{
  /* Grab the arguments of CSS_polar2x(). They are in the format: type,name,argument#
*/
    CSS_ARG(float,r,0); /* argument #0:  type=float, name=r, number=0 (1st)       */
    CSS_ARG(float,theta,1); /* augument #1:  type=float, name=theta, number=1 (2nd)
*/
    double rad_per_deg=.01745329;
  /* use the CSS_polar2x() arguments as the arguments of cos().*/
    return(r * cos(theta * rad_per_deg));
}
```

The second step is to add a function declaration to the statefns.h file. For the example above, the entry looks like this:

```
   float CSS_polar2x(CSS_ARGS)
```

The third step is to add an entry into the SystemFns[] array in the ufnlst.c file. SystemFns[] = {} lists the master program's system functions which are accessible from CSS. Since CSS can't pass arguments directly to the stack, it is necessary to write an interface function for each of the master program's functions that one wishes to access. Using the example of polar2x:

{ "polar2x", 0, "fff", CSS_polar2x }, the fields specify the following:

- The string to be recognized by CSS. That is, "polar2x" is the name that you would specify in your timing file to call this function.
- 0 (will be translated into the index within the array for internal use)
- the syntax string: "fff".

This cryptic code identifies variable types for the function and its parameters. The first "f" is for the function return type (in this case "f" stands for float). This, and only this, position can also take the value 'v', for void. The second "f" is for argument #0 (see above), the third "f" is for argument #1, etc. The syntax string must contain at least one entry: the return value of the function. "v" is only valid as a return value. Functions which take no arguments, but return something (say float) would have a syntax string of "f", NOT "fv". Also, functions with no arguments take a syntax string of length one, such as "v", NOT "vv". The other types are:

c	character
i	integer
l	long
f	float
C	pointer to character
I	pointer to integer
L	pointer to long
F	pointer to float
0	double pointer to character
1	double pointer to integer
2	double pointer to long
3	double pointer to float
.	variable length argument list
v	void
V	pointer to void
P	pSysfn (pointer to a system function)
Q	pCSSfn - the starting index in the array m.code.a (used in the CSS)

Example:
void MessageString(int which, pchar message) would be characterized by "viC"

The '.' is used for variable length argument lists, such as:
void printf(pchar fstring, ...) characterized by "vC."

Thus, when '.' is the last entry in a syntax string, that argument is optional, and any number of arguments of any type may follow the last required argument. CSS ensures that the proper number of arguments are popped off the internal stack. Only the last entry in a syntax string may be '.'.

- CSS_polar2x is the function name that will actually be called internally from CSS.

8.4.2 Adding library functions

For those functions that are direct ports of internal ones, like C library math routines, there is a simplified (and faster) interface. For example, log() is of the form

```
double log(double);
```

In order to easily add log to CSS, add the following three lines:

1) In statefns.h, add:
```
CSS_1d(log,float,float)         // NOTE:  does not end in ';'
```
   i. where the first argument (log) is the name of the library function, the second argument (float) is the type of the return value of the function, and any subsequent arguments (float) are the types of the input parameters to the function.

2) In a source code file, such as \ufns_system.c, add:
```
CSS_1(log,float,float)             // NOTE:  does not end in ';'
```
   i. Note that although log() expects arguments of type double, this interface will work, since the float passed to it will be promoted to double, and the result will be truncated to float by the C compiler.

3) In the SystemFns[] array in the file \ufns\ufnlst.c, add:
```
CSS_1s(log,float,float)         // NOTE:  does not end in ';'
```

There are CSS_?(), CSS_?d(), and CSS_?s() macros for functions passing up to 15 arguments (and more can be made). Note that macros specify a return value and the arguments, and '?' needs to be correct, or else the macro will balk (don't worry, it will let you know). Microsoft's C/C++ compiler does more rigorous error checking, and will warn when a void function returns a value. This is not an error, since void functions traditionally return an integer value that is ignored, and those values are returned in a register, as opposed to on the stack, so the stack is not corrupted. However, in order to eliminate those warning messages, an additional class of CSS_?v() macros have been added, which expand to

```
void CSS_????(dataEls *args) {
           ????(...);
       }
```

instead of trying to return a value. For example, the way to map the function free() would be

```
CSS_1d(free,void,pchar)
CSS_1s(free,void,pchar)
CSS_1v(free,void,pchar)
```

Notice that the macros that end in "d" and "s" remain the same. Only the one that translates into code needs the "v" suffix.

8.4.3 Adding External Variables to Cortex

In the file \ufns\ufnlst.c, EXTERN_VAR ExternVars[] = { } lists all of the variables intrinsic to the master program which are accessible within CSS. For examples, follow the prototype listed in ufnlst.c. One major caveat: Arrays

are treated differently in C depending upon whether they are statically or dynamically defined. The latter include both a pointer (4 bytes) and allocated space for the elements. To conserve space, C compiles static arrays so as to remove the need for the pointer. Thus, one might think of declaring EV("array",pint,&array[0]);. This may not be completely safe (and EV("array",pint,array); is definitely fatal). The best approach is to actually have a separate variable that points to the static array. For example:

```
int    arrayData[10];
pint    array = &arrayData[0];
EV("array",pint,array);
```

As to access of these variables, the fact that they are listed in ExternVars[] makes them globally accessible in CSS, so it is not necessary to have a line like:

```
extern int array;
```

and in fact, CSS will generate error messages or worse if this occurs.

8.4.4 Rebuilding Cortex after adding new functions and variables

After new functions or variables have been added, the Cortex executables must be rebuilt. If your new functions are in a new file, you will need to modify makefile in the \source directory to include your new source file name before rebuilding. If you used _userfns.c, then makefile does not need modification.

The argument-passing scheme may be changed in the future. Therefore, it is essential that the macros be used instead of their translations.

Linking cortexcc.exe generates a slew of error messages. It is supposed to, since the file containing SystemFns[] contains pointers to code. Neither that code, nor the pointers needs to be used by the parser. Therefore, any unresolved externals from that file are OK and expected, and cortexcc.exe will run perfectly correctly. Unresolved externals in the master program are unacceptable and will cause the system to crash.

9. Data File Structure

As described earlier in this document, the disk file consists of the sequential data for each trial. Each trial's worth of data is preceded by a header. The structure of the header is given in the file disk.h, which is typically in the \source\include directory. The structure is as follows:

```
 typedef struct {
   unsigned short   length;          /*not currently used*/
   short            cond_no;         /* the experimental condition numb.*/
   unsigned short   repeat_no,       /*repeat # of entire block structure*/
                    block_no,        /*block no., this condition*/
                    trial_no;        /*trial # for this condition/
   unsigned short   isi_size,        /*size in bytes of this trial's array of
                                     times*/
                    code_size,       /*size in bytes of this trial's array of
                                     codes*/
                    eog_size,        /*size in bytes of this trial's array of
                                     eog values*/
                    epp_size;        /*size in bytes of this trial's array of
                                     epp values*/
   unsigned char    eye_storage_rate /* millisecs between eye samples */
   unsigned char    kHz_resolution;  /*0/1 if 1 msec, 10 if 0.1 msec*/
   short            expected response;
   short            response;
   short            response_error;
} DIR_ENTRY;
```

All but two of the above values are short integers. The types of response errors are given as an enumerated type in disk.h (see RESPONSE_ERROR). Following the header are the time and code arrays, the epp array, and then the eye movement array. For each event code (such as the occurrence of a spike, the onset of a stimulus, etc.,) there is an associated time. The time is measured in milliseconds from the start of the trial, and is given as a unsigned long integer (i.e. 32 bits). The event codes are listed in histogra.h as an enumerated type (see CODE).

Note: for repeat #, block #, cond. #, etc, all values are base 0 (e.g. the first trial is trial 0).

repeat_no. The repeat # for the entire block structure. The user sets up how many conditions per block and how many trials per block. When all of the blocks have been run once, the repeat number is incremented.

trial_no. Trial number for this condition.

block_no. Conditions can be assigned to blocks. For each block, the user specifies the number of trials and conditions to run before moving on to the next block. This is the block in which this condition is located.

expected_response. This field gets set during the trial from the value under the TRIAL_TYPE heading of the conditions file. See DISK.H for a list of the TRIAL_TYPEs.

response. This field gets the value that you pass into one of the response_???() functions. For example, if you called response_correct(15), then "response" would contain the value 15.

response_error. This field gets the value that one of the response_???() functions return. For example, response_correct() returns NO_ERROR, which is 0.

9.1 Sample data file

Each data item occurs sequentially in the file.

The following header data items each occupy two bytes (one integer) in the data file, except for eye_storage_rate and kHz_resolution, which each occupy one byte.

DATA	MEANING
0	length (not used)
1	condition number
0	repeat_no,
0	block number /*block no, this condition*/
0	trial_no
2560	size of array of time codes (e.g. 2560 bytes, 4 bytes per code)
1280	size of array of event codes (e.g. 1280 bytes, 2 bytes per code)
500	size of array of eog values (e.g., 500 bytes, 2 bytes per value)
500	size of array of evoked potential values (e.g., 500 bytes, 2 bytes per value)
4	milliseconds between eye samples - e.g. 4 = 250 Hz (BYTE)
0	time resolution of interrupt routine (1 kHz) (BYTE)
1	code for expected response
2	code for actual response
0	code for response error

The following data items each occupy 4 bytes (long integer)

0	first time code (see associated event codes below)
100	second time code
500	third time code
etc.	

The following data items each occupy 2 bytes (short integer)

18	first event code
24	second event code
40	third event code
etc.	

The following data items each occupy 2 bytes (short integer)

15	first epp_x value
30	first epp_y value
18	second epp_x value
35	second epp_y value
etc.	

The following data items each occupy 2 bytes (short integer)

480	first eog_x value
220	first eog_y value
485	second eog_x value
220	second eog_y value
etc.	

Following these data items would be the header for the next trial's worth of data...and so on until end of file.

10. Troubleshooting

10.1 Mouse doesn't work in PLAY mode, in timing file execution, or in touch screen emulation mode

The mouse driver must be loaded in DOS. Check your autoexec.bat file. It should contain a line such as: c:\mouse\mouse.exe (of course, this depends on the location and name of the mouse driver). After that is added, reboot your computer. Now, quickly test if the mouse works in DOS mode at all, by testing it in the EDIT program in DOS. If that doesn't work, then the mouse will not work in Cortex either.

10.2 Send/Receive computers cannot communicate: Debugging the serial connection

1) Check that you have the correct serial cable (see section 2.1.8), connected between the two computers, and it is between the correct COM ports. That is, on some computers there are two COM ports, so make sure that the cable is between COM1 on both (the default).
2) Are the COM ports enabled and OK? Boot up the computers into Windows, and look at the properties of the COM ports in the Control Panel:System:DeviceManager window. Make sure that it says that the ports are operating and there are no conflicts. There should be something in the system BIOS that lets you enable/disable the ports. If you can find out how to boot up into the BIOS/CMOS settings, you can check to make sure that the ports are enabled. (Usually, you have to do something like CTRL-ALT-ENTER, or F2 during reboot, etc., and it will show the CMOS/BIOS settings.)

   You can also determine whether GreenLeaf CommLib (the communications library that Cortex uses) can recognize the port, by using the debug commands suggested in their product documentation.

   "From a DOS command prompt, start debug.exe:

```
C>DEBUG
-i2fa
01
-i3fa
01
-q
C>
```

   This procedure will look at the Interrupt ID Register for COM1 and COM2, respectively. Each command reads the contents of one of the UART registers for COM1 and COM2. The '01' returns mean there is a UART present. If you see anything other than '01' or 'C1', it means that CommLib will not recognize the port. 'C1' means the hardware is a 16550 or related UART and the FIFOS were left enabled. Some applications will experience difficulty detecting a port if a prior process did not disable the FIFOs."
3) Are the computers capable of a baud rate of 115200? As in 2) above, look in the Port Settings panel. If you can not scroll down and see 115200 listed in the "bits per second" field, then your computer can not handle 115200. You will need to change the COM_PORT line in your cortex.cfg file on the send computer to the highest value possible for your computer. You will also need to start up the receive program, so that the baud rate is supplied as a command line argument. For rvesactx.exe, type: rvesactx @1200 to set its baud rate to 1200. There is no space between the @ and the 1200. For the DirectX receive program, put the @1200 into the shortcut properties at the end of the Target: line, for the shortcut to the dxrecv.exe program that you created on the desktop.
4) Check if there are any .err files generated in the \remote32 directories on both computers. These files might give some hints. Make sure that you check the date of the .err file. It is overwritten only if there are errors, so it might contain text that is not even relevant to the current problem. To be sure, you can just delete the .err file on the receive computer. It will be generated again if there are errors during execution.

5) Make sure that the computers are booted up in true DOS, by hitting F8 while it is booting up, and choosing Command Prompt Only. (If you are using DirectX receive, only the send computer needs to do this.)
6) Another possible scenario is that both sides of Cortex start up properly, but the process stops before displaying the menu. That is, the screen freezes without any error message with the text "using font HELVB, FON ...". If this is the case, then it is not a communications problem. Cortex got past the part of the code that does the "serial handshake" between the two computers. It is probably a problem with the initialization of the graphics. Make sure that you have the appropriate graphics driver installed (Scitech or DirectX), and the driver is operating properly. Also check that the resolution that you have specified in the GRAPHICS_SPECS line of your cortex.cfg file on the send computer, will actually work with the driver that you are using.

10.3 Image file does not display

Review section 7.1 of this manual, and make sure you are using an image format that is supported by the version of Cortex that you are using. Remember also that the 3-letter extension of the image file name is important in telling Cortex what image file format to expect.

If you are using the two-computer versions of Cortex, it is imperative that the image file resides on the receive computer in the directory which you have specified in the items file. If you have not specified a directory, Cortex will look for the file in the directory from which you are running the receive program. If Cortex can not find the file, it will report the error "can't calc item size" on the send computer screen.

Another common problem is that images created in other graphics programs do not display properly in Cortex at 8 bpp. To solve this, when using Adobe PhotoShop, for example, to create the .BMP file, make sure to save the BMP files as "Indexed Color" and "8 bits per pixel". When you save the file as a bmp file in PhotoShop, it prompts you to choose between Windows and OS/2 format. Make sure it is Windows format, not OS/2.

10.4 Movie file does not display

Review section 7.2 of this manual, and make sure you are using a movie format that is supported by the version of Cortex that you are using. (That is, only the DirectX receive program will support movie types other than the original Cortex movie format.) Remember also that the 3-letter extension of the movie file name is important in telling Cortex what movie file format to expect. Also remember that not all video codecs are supported by DirectX, so check the list of supported video codecs on the Microsoft web site.

As in 10.3 above, if you are using the two-computer versions of Cortex, it is imperative that the movie file is on the receive computer in the directory which you have specified in the items file. If you have not specified a directory, Cortex will look for the file in the directory from which you are running the receive program. If Cortex can not find the file, it will report the error "can't calc item size" on the send computer screen.

If you are using the original Cortex image format movie, make sure that the number of frames in the movie is specified in the header of the first frame as (number of frames minus 1).

10.5 Sound does not play

Sound is only supported for the two-computer versions of Cortex. If you are using a two-computer version, then the sound card must be installed and operational in the receive computer. Additionally, the sound (.wav) files must be located on the receive computer in the directory which was specified in the SOUNDload() function call in your timing file. If no directory was specified, then Cortex will look for the sound file in the directory from which the receive program is running.

For the Scitech (MGL) receive program, the sound card must be SoundBlaster compatible. For the DirectX receive program, the sound card must be supported by Microsoft DirectX.

10.6 Timing file does not compile

Sometimes when you load a timing file into Cortex, it will not compile properly and will produce error messages. The Cortex compiler gives very cryptic error messages, but usually the line numbers will give you a hint as to where to look in your file. For example, you may get error messages that look like this:

```
PREPPY [encodes.h(107)][warn1]redefining macro "NONDISPLAYABLE ENCODES.
YACC[memory.1(6)][ERR1]@"appear":parse error
```

The number in parentheses on the YACC line, i.e. (6), points to the line in you timing file that is having a problem. In this case, when I looked in the file, I saw that the word "appear" started on the line without comment (//) symbols. If you cannot see anything wrong with the line specified in parentheses, look at the line above it. If you can not find the error, try compiling your program with a standard C compiler. Of course, a C compiler will generate linker errors since it will not be able to find the Cortex functions, but it should also find syntax errors.

Do not worry about the PREPPY encodes.h(107) error message. It should go away when the errors in your timing file are fixed.

10.7 The two-computer version of Cortex crashes during execution

This problem is usually due to not running the computers in true DOS mode. See the troubleshooting tip in Section 10.2 at step 5.

10.8 Spikes are not appearing in the histograms

First, you must determine if you just do not see them on the screen, or that they are not in the data file. Check the data file with one of your Cortex data analysis programs or Cortex Windows Workbench to see if any spikes are being recorded.

If the spikes are in the data file, but are just not showing up on the screen, then it could be the ordering of the function calls in the timing file. That is, one of the response_???() functions and end_trial() must be called *before* update_histogram(); otherwise, the histogram will not be updated properly. (Also, make sure that the function collect_data(ON) is being called to initiate spike collection!)

Another thing to check is whether or not the trials have been labelled as "errors" by the response_???() functions. By default, trials which are "error trials" do not have their spikes accumulated during end_trial(). You can override this default behavior, and can actually have these spikes included in the histogram. There is an option in the Run:Parameters:General menu which is called "Include error trials in histograms". If you set this parameter to "Yes", then the error trials will be accumulated in the histograms, too.

If there are no spikes in the data file, there are a few hardware things to check. First, the spike inputs should be low-going-high. Second, make sure that the spike latch circuitry is really correct. There are some circuit diagrams up on the Cortex web site. Third, make sure that the ribbon cable is attached to the correct data acquisition board connector. (The CIO-DAS16 board is confusing, because there is also a connector on the "face" of the board, besides the one that is at the back of the computer.)

Finally, check the Cortex.cfg file, to make sure that it is set up properly for the data acquisition board that you are using. That is, you must have the correct parameters for the DEVICE, THREAD, and MULTI_SPIKE lines of the Cortex.cfg.

10.9 Colors do not look correct for BMP files

If you want to change the LUT for an image, it is almost mandatory to use CTX files. The other types of images (i.e., .BMP, .JPG, and .PCX) keep their LUT internal to the image file. For non-.CTX file cases, you would need to know the internal structure of the image file and where the individual pixels were located. Even then, it may not work to change the Cortex LUT, and expect the bitmap color to change.

In the DirectX receive program, .bmp files in 8 bpp will use the Cortex LUT. (For example, if the LUT is not in the conditions file, then the image will be displayed in black and white.) It is therefore recommended to create a default LUT which contains the Windows system palette, for use in 8 bpp. At greater than 8 bpp, the images will look fine, and will not need/use the Cortex LUT. There is a utility program, bmp2lut.exe, which extracts the LUT from a .bmp file. It is available "as-is" from the Cortex ftp site.

In the Scitech receive program, non-.CTX files use their own internal LUT, so a Cortex LUT does not need to be specified in the conditions file. Note that if two image files are to be displayed at the same time, their LUTs must match, or the colors will look bad. Scitech will use the palette from the non-.CTX file, even if one is provided in the Cortex conditions file. It may therefore, influence the colors of other items.

10.10 Display of movies is slower than the refresh rate of the monitor/graphics card

The rate at which movies can be displayed depends upon many things. That is, it depends on the refresh rate of the monitor, the refresh rate capabilities of the graphics card, the speed of the CPU, and the size and complexity of the image that you are trying to display. The refresh rate is the number of times per second that the monitor refreshes the screen. However, if the graphics card is not able to process the image within one refresh of the monitor, the image will not be able to be redrawn at the refresh rate of the monitor.

Neither the Scitech receive program nor the DirectX receive program is able to dynamically set the refresh rate for all graphics cards. Therefore, it is usually necessary for you to manually set the refresh rate inside Windows, and within the Scitech Control Panel (if you are using Scitech). Once the refresh rate is set, you must also specify the refresh rate in the GRAPHICS_SPECS line of the Cortex.cfg file on the send computer.

If you are not achieving the speed that is necessary, you will have to either change your hardware or your movies. That is, you will need to get a faster graphics card, monitor, or computer. Or, you will need to alter your movies so that they are smaller in size and use 8 bit-per-pixel color depth.

10.11 EOG data is not being stored to file

Make sure that the wiring to the data acquisition board is set up, so that Channel 3 is the x value of the EOG, and Channel 4 is the y value of the EOG. (See Section 3.3.3 and 3.3.4)

The timing file must contain a call to the function put_eye_data_in_buf(ON), to start EOG data collection. To end EOG data collection, call this function with "OFF" as the parameter. The EOG data does not get stored with timestamps. Therefore, the usual strategy is to call the function encode(), near the call to the put_eye_data_in_buf(ON) function, to place a timestamped code into the data file.

The rate at which EOG data is stored is controlled through the Cortex menus. In the Run:Parameters:General:EOG_storage_rate (# of clock ticks between entries) field, a value greater than 0 should be entered. This value is used as a countdown timer that determines when the x and y EOG values are stored. Since only one analog value is read per millisecond, this value should be set to at least 2 clock ticks between entries.

So that the values will actually be stored to file, you must also set the "Save eye data to file?" parameter to "yes". This parameter can be found in the Run:Parameters:General menu.

10.12 EPP data is not being stored to file

Cortex is able to input 4 channels of analog data. If you are using a version before 5.3, Cortex could only read/store two analog channels. If you are using a version of Cortex after 5.3, you will be able to store all 4 analog channels. To use Cortex with 4 analog channels:

1. Wire the DAS16 portion of your data acquisition card so that channels 3 and 4 are used for the EOG data, and channels 1 and 2 are used for the EPP data. Refer to the diagram in Section 3.3.3 and 3.3.4 for information on how to do this.

2. Add the line "A2D_CHANNELS 4" to cortex.cfg. This will make CORTEX cycle among the 4 analog channels, reading one per clock interrupt.
3. The timing functions put_epp_data_in_buf(ON) and put_eye_data_in_buf(ON), respectively, must be called before data can be stored in the respective buffers. To end EOG and EPP data collection, call these functions with "OFF" as the parameter. The EPP data does not get stored with timestamps. Therefore, the usual strategy is to call the function encode(), near the call to the put_epp_data_in_buf(ON) function, to place an timestamped code into the data file.
4. Params:General:EPP_storage_rate and Params:General:EOG_storage_rate control the rate at which data is stored into the buffers. These values are used as countdown timers that determine when the x and y EOG/EPP values are stored. Since only one analog value is read per millisecond, this value should be set to at least 4 when storing all 4 channels.

10.13 Adjusting the gain of the analog data

First, make sure that the data acquisition board and your analog input have the appropriate settings. Cortex expects the analog data to be input as bipolar (+/- 5V) data. Additionally, the wiring is usually configured as 16 single-ended channels. On the ISA-bus data acquisition cards (DASH-16, CIO-DAS16, CIO-AD16, and CIO-DAS1602/12), these settings can be made with jumpers on the card. Refer to the manual that was shipped with the data acquisition board for more information about the jumpers. On the PCI-bus data acquisition card (PCI-DAS1602/12), there are no jumpers on the board. Therefore, the settings for these features can be accomplished via the SINGLE/DIFFERENTIAL and BIPOLAR/UNIPOLAR parameters on the DEVICE line in the Cortex.cfg file. By default, the PCI-DAS1602/12 is initialized by Cortex to be BIPOLAR and SINGLE-ENDED.

Once the hardware is set up properly, there are two places in Cortex where the gain of the analog data channels can be set. The first place is in the A2D_GAIN line in the Cortex.cfg. The A2D_GAIN is used to directly program register BASEADDR+11 (see the manual that was shipped with your data acquisition board) to set the gain control. For example, with the CIO-DAS1602/12, if the A2D_GAIN is set to 1, the board should be able to handle +/- 5V.

The second place that can control the gain is in the Run:Parameters:General menu of Cortex. The "eog_gain" in the Run:Parameters:General is used in the Cortex code as a multiplier to the EOG values that are read from the board. After the EOG value is read from the board, it is altered by the values for the offset and gain that you set in Run:Parameters:General as follows:

```
EOGnew_x = (EOG_ENTRY) ((raw_eog -params.eog_xoffset) * params.eog_gain);
```

There is a similar calculation for EOGnew_y.

The EOG gain from the Run:Parameters menu is NOT applied to the EPP values (just the EOG values). On the other hand, the A2D_GAIN value is used for all the analog channels, including the EPP.

10.14 Can not see spike codes when viewing data

Cortex reads the encodes.h file in the directory from which you run Cortex (and Cortview.exe) to generate the mapping between encode() values and the values printed from Run:Display menu option from within Cortex, or from the standalone DOS program Cortview.exe. Cortex parses the encode values, expecting a format like

```
#define SPIKE1 1   // optional comments
```

Two other #defines are reserved, each of which can be used multiple times:

```
#define DISPLAYABLE_ENCODES
#define NONDISPLAYABLE_ENCODES
```

Values that have #define lines after the "#define DISPLAYABLE_ENCODES" line will be visible from Run:Display, whereas values after "#define NONDISPLAYABLE_ENCODES" will not. This way, one can specify high spike values (like #define SPIKE3 1000), or other user-defined values, and choose whether or not to have them clutter up the Run:Display menu.

10.15 "ERROR: CODE_ISI buffer overflowed *x* times (data was lost)" on the user screen

The CODE_ISI buffer holds both the spikes, and the values generated by calling the encode() function in the timing file. The error message means that more spikes or encodes are being generated than the Cortex output data file has been configured to hold.

The sizes of the CODE_ISI, EPP, and EOG buffers are specified in the DATA_STRUCTS line of the Cortex.cfg file. There are some comments in the cortex.cfg file:

```
// DATA_STRUCTS <#CODE_ISI_els> <#EPP_els> <#EOG_els> [all as max per trial]

// <#CODE_ISIels>:  The max number of encode (CODE and timestamp) elements
//      which can be stored per trial.
```

In the Cortex.cfg file that is shipped with Cortex, the <CODE_ISI_els> is about 4000. In other words, it can only store a total of 4000 spikes and encodes. If Cortex is producing an error that the buffer overflowed, you are going to need to increase the buffer size in the Cortex.cfg file by at least the amount that it reportedly overflowed. The CODE_ISI_els value can be set to about 100000 or more. However, since the buffers are created from DOS memory, Cortex will not be able to start up if the buffer sizes are too large.

In any case, before the increasing the size of the buffer, you should look at the Cortex output data file, to determine if it is encodes or spikes that are filling up the buffer. Maybe, in the timing file, there is an encode() statement inside a loop and it is writing out thousands of values into the buffer.

Otherwise, if the buffer is not being filled with encodes, then it is being filled with spikes. If you know that there should not be that many spikes coming into Cortex, then it is most likely a hardware problem. For example, if the spike inputs are not being latched properly, or are not being reset each millisecond, then you will get a huge amount of false spike data. Check that all the spike input wires, the reset, and ground are all wired properly. In particular, make sure that all the lines that are specified in the MULTI_SPIKE line of the Cortex.cfg file are actually wired to a spike source.

Appendix A. Technical Notes on Blocking

A.1 Overview of the built-in capabilities for general staircasing designs

The user can create several blocks of conditions, essentially a block of blocks. The default criteria for finishing a block, is that the specified total number of conditions has run. The criterion for finishing the block of blocks is that the desired number of blocks has completed. Additionally, one can set a repeat counter to repeat the entire block of blocks again.

The menus let the user have fine control over the randomization of the conditions and/or blocks. For example, the user can guarantee that the trials are randomly run 10 times, but that the same condition is never repeated more than twice in a row. Similarly, the settings can make it possible for there to be up to N repeats of the same condition in a row. Of course, it is also possible to run the conditions/blocks incrementally.

Although there are many available parameters, the user only has to enter 5 of them per block. In fact, unless the user changes the Menu_Length option in the Sizing menu, the additional 9 parameters are not accessible.

More complex completion criteria for the individual blocks can also be easily specified. For example, it is possible to run a block until the overall percent correct of its conditions exceeds some threshold. Or, one might only want to look at the most recent 20 (or so) trials and define completion as success on 18 of them.

CORTEX also gives flexibility in the handling of error conditions. Parameters can be set to guarantee that a certain number of correct trials of each condition are run, so the error conditions would have to be repeated. Choices include ignoring error conditions, repeating them immediately, or repeating them later. If a block is considered done based on recent performance or percent correct, then a delayed repeat would effectively increase the likelihood that that condition is selected from the remaining group.

Additionally, it is possible to abort the block if the subject's performance deteriorates too much. One can specify that the block is an error if there are too many error trials (an absolute number), if the percent correct drifts too low, or if the number of recent correct trials drifts too low. The user can opt to repeat a block if it is an error, or just skip on to the next one.

Finally, the user has access to some of the internally stored blocking parameters and functions, which are necessary when designing a Manual staircase design.

A.2 Block/Repeat Menu

The Block/Repeat Menu can be reached from the Run:Parameters menu in Cortex. Blocking has six sub-menus associated with it:

Sizing **Master block** **Individual blocks** **Save blocks** **Restore blocks** **Clear**

A.2.1 Sizing

1. **# of Blocks**: The number of distinct blocks that should be created. The default is one block.
2. **Menu_Length**: This parameter determines the lengths of the Master Block and Individual Blocks menus. The options are:
   a. Minimal: presents 5 parameters
   b. NoErrorBlocks: presents all options except for 4 criteria which can abort the Block early
   c. Full: presents all options.
3. **Length of circular buffers**: These buffers contain one-character codes representing the result of each trial.

A.2.2 Master Block

1. **# of Repeats**: The number of times that the blocking structure should be repeated.
2. **Block Order**: Specifies the order in which the blocks should be run. The choices are as follows:
   a. **Incremental**: The blocks to be run will follow in sequential order (i.e. 1,2,3...). If Delayed_Retry is selected for On_Error, there may be discontinuities (e.g. ...8,9,3,7) if the blocks which need to be repeated have lower block numbers than those which would normally be at the end of the sequential list.
   b. **Rand_w/o_replacement:** Guarantees an even distribution of the number of blocks over the maximum number of blocks, but randomizes choice of presentation.
   c. **Rand_with_replacement**: The blocks will be run in completely random order.
3. **Max blocks, first block, and last block**: Specifies the total number of blocks, and the first and last block number of the range of that blocks that should be run.

The next option does not appear unless Menu Length was set to Full or NoErrorBlocks in the Sizing submenu.

4. **clear Block/Cond circ_buf when?**: These refer to the automatic resetting of the circular buffers (recency tables) and the num_correct and num_errors counters they contain. Although each is updated correctly, CORTEX uses the BLOCKstats (not the CONDstats) to determine the order of blocks and conditions for the next trial. The default for clear_cond_cbuf_when is "at_Start". The default for clear_block_cbuf_when is "when_Change_block".
   a. **Manual**: CORTEX never clears BLOCKstats or CONDstats, so the user must do so manually.
   b. **at_Start**: Cortex clears statistics when Run:Start is chosen, but not upon Run:Resume.
   c. **when_Change_block**: Cortex clears statistics when a block (correct OR error) has finished.
   d. **when_Repeat_all_blocks**: Cortex clears statistics when the repeat counter is incremented.

The following options do not appear unless Menu Length was set to Full in the Sizing submenu.

5. **Next_block** and **Next_cond**: When these values are non-zero, these values override ALL other options. The only restriction is that valid block and condition numbers must be specified. The values start from 1 (not zero).
6. **On_error?:** Specifies what to do if errors occur. The choices are: Ignore, Immediate_Retry, and Delayed_Retry.
7. **Max_errors**: Default is 10.

A.2.3 Individual_blocks

1. **Condition order?:** Specifies the order in which the conditions inside the block are to be run. The choices are as follows:
   a. **Incremental:** The conditions to be run will follow in sequential order (i.e. 1,2,3...). If Delayed_Retry is selected for On_Error, there may be discontinuities (e.g. ...8,9,3,7) if the conditions which need to be repeated have lower cond_ids than those which would normally be at the end of the sequential list.
   b. **Rand_w/o_replacement:** Guarantees an even distribution of the number of conditions over the maximum number of trials, but randomizes choice of presentation. Thus, if 25 trials of conditions 1-10, each will be run at least 2 times, but 5 of them will be run 3 times. Cortex figures out the number of times that each condition should be presented over the maximum number of trials. Then, the conditions can be presented in any order. It will only be evenly distributed if you run all of the trials.
   c. **Rand_with_replacement:** The order is completely random. Thus, if 25 trials of conditions 1-10, there will be 25 trials specified, but it is not uncommon for there to be 5 of one trial, and 0 of several of them.
2. **On error?**:
   a. **Ignore**: Do not do anything special when errors occur; count them as normal trials. Thus, if you specify Max_trials = 25, there will be exactly 25 trials in that block. If On_error is not set to "Ignore", error trials are not counted towards Max_trials, so Max_trials becomes "Max_corrects."
   b. **Immediate_Retry**: Immediately repeat the same condition, if an error was made. This will also increment the retry counter. If you set Max_retries to something other than 0, then the block can be aborted as an error if the same condition is repeated too many times in a row.

c. **Delayed_Retry**: Retry the condition at a later time in the block, if an error was made. If Condition Order is Incremental, then the condition will be done at the end of block (assuming using ==MaxTrials as Done_when?).

3. **Max trials, first condition and last condition**: Specifies the total number of trials that should be run, and the condition number of the first and last condition in the set.
4. **What defines Done**?
   a. **==MaxTrials**: as soon as Max_trials have occurred, the block will be completed. If On_error? was set to Ignore, this will be number of trials; otherwise, it will be number of correct conditions.
   b. **%Correct>=minPctOK && >= minTrials**: Requires that you set minPctOK and minTrials variables below. Once at least minTrials have occurred, Cortex checks to see whether percent correct >= minPctOK, where percent correct is computed from BLOCKstats. (The num_correct and num_errors are stored in the circ_buf array. When you reset the circ_buf array, you also reset these two params). If MaxTrials is low, you may exceed it before the percent correct condition is satisfied. If so, the block is reset, but the BLOCKstats is not. Thus, if you had specified 25 trials of conditions 1-10 and Incremental, you would get an even more lopsided distribution.
   c. **>=recentOK of recentDone && >=minTrials**: Requires that you set these three parameters. This condition examines the BLOCK circular buffer. (As mentioned above, none of these options looks at the COND circular buffers, or even the COND pct_correct parameters. These are available for MANUAL selection, however.) RecentDone must be <= circ_buf_size. Cortex will not let you specify a larger value. The reason that it is used at all instead of just forcing RecentDone to be the same as circ_buf_size is that some blocks might want to access differently sized subsets of the circ buf, especially for training paradigms. RecentOK is the number of correct trials within recentDone. For example, if you want the trial to continue until the subject gets the most recent 90% of the trials done, you might specify 9 for recentOK and 10 for recentDone (or 18/20,...). minTrials is an optional parameter in case you want minTrials > recentDone. As with (b), if maxTrials set too low, the block will continue to be re-initialized until the correct (or error) conditions are met.
5. **minPctOK**: Specifies the minimal percent correct threshold. That is, if the block's current percent correct is higher than minPctOK, then the block will be considered correct.
6. **minTrials**: Specifies the number of trials that the block will run before testing other parameters such as minPctOK.
7. **recentDone**: Specifies the number of the most recent trials that will be used to measure the subject's progress using either recentOK or recentOKtooLow.
8. **recentOK**: Specifies the number of correct trials required from the block's most recentDone trials before the block will be considered correct.
9. **Max_errors**: Specifies the maximum number of errors allowed before the block is considered aborted. For longer trials, you may want to automatically abort if too many error trials occur. If Max_errors value is not zero, then as soon as this many errors occur, the block will aborted and considered an error. The master block parameters are then queried, and can be specified to have the entire block be repeated, for example. Further, if one sets clear_block_cbuf_when to something other than Start or Manual, all of the statistics for that block will be cleared first (so entire block will be re-done). Otherwise, only as much of the block is re-done to determine the new correct or error status of block. This only makes a difference in case of Max_retries, since that is stored locally and is always reset when a block is re-started. (Max_errors will not be cleared in this latter case, so if the block aborted because of too many errors, it will still be considered in error unless it is reset, so NO additional trials will be run).
10. **Max_retries**: Specifies the maximum number of retries allowed when the On_Error variable is set to Immediate_retry. This parameter is always reset any time a block is re-initialized. If it is set to 0, then it has no effect.
11. **recentOKtooLow**: If the block's most recentDone trials have this many correct trials or less, the block will be aborted. If this is non-zero, then it behaves like a low percent_corrects cuttoff based upon the minTrials and recentDone params (see above for recentOK). For example, after giving the subject 40 trials to get used to the procedure, you might want the block to be considered correct as soon as the subject gets 18 of the last 20 trials correct. On the other hand, if the subject is getting totally lost, you might want the block to be stopped (i.e. if less than 13 of last 20 were correct if it is a binary choice).

12. **PctOKtooLow:** If the block's percent correct becomes this low or lower, the block will be aborted. Allows the block to be aborted if minPctOK is ever less than PctOKtooLow after minTrials have passed. One could then opt for a delayed retry of that block (from the MasterBlock menu) if desired.

A.2.4 Save Blocks

Saves the blocking parameters into a file.

A.2.5 Restore Blocks

Restores the blocking parameters from a file.

A.2.6 Clear

Clears the CONDstats, BLOCKstats, and/or re-initializes the settings to one block.

A.3 Functions available for MANUAL operation

1. **BLOCKset_next**: sets the next block and/or condition value. You must specify non-0 to change the block/condition, or specify 0 to stay the same. If BLOCKset_next(-1,-1) is called, this triggers the end of the block of blocks. The one limitation is that the user must specify valid block and condition numbers.
2. **BLOCKget_block_num** and **BLOCKget_cond_num**: returns the current block and condition numbers.
3. **BLOCKget_max_vals**: returns the maximum values for block and condition. The minimum value is 1. This way the user does not have to know how many blocks and conditions there are ahead of time.
4. **BLOCKget_pct_correct**: returns the percent correct for a given block or condition.
5. **BLOCKget_stats**: returns the total number of trials, the number of correct trials, and the recency (circular) buffer for a given block or condition. When the general blocking menus look at recent correct, they are always looking at the block recency buffer. There are also recency buffers for conditions, so one could drop a particularly problematic condition out of a block if so desired.
6. **BLOCKclear_stats**: Allows the user to clear the recency buffer and percent correct (num_trials and num_correct) tables for a given block or condition. Can also do this for all blocks and/or conditions.
7. **BLOCKget/set_control_info**: Allows the user to view or change many of the control parameters for individual blocks. Allows semi-manual operation. For example, early on in a trial, one might want to let the subject make many mistakes. Later in a testing session, though, one might want to abort the block if the accuracy drifts too low. These functions let these criteria be changed mid-block.

A.4 Technical details on the internal operations of the Cortex code

The file Runtrial.c contains code which loops on the variable BLOCKglobal.repeat_counter. There is an infinite while loop which calls BLOCKnext_cond(). When BLOCKglobal.command == BLOCK_DONE, all of the blocks are done, so the while loop as broken, and the repeat_counter is incremented. If an internal error occurs, BLOCKnext_cond() will return 0, and "Invalid cond" will be printed.

BLOCKnext_cond() calls BLOCKwhich_block(), which initializes the Master Block if necessary, and returns the current block number. It also checks to see whether correct/error conditions have been met for the Master Block, in which case it returns BLOCK_DONE. BLOCKnext_cond() checks the results of the last condition. (If it was an error, special processing may be necessary.) It checks to see if Max_errors or Max_retries were met (failure conditions). It checks to see if any of the correct trial conditions were met (but failure takes precedence). If a block was completed (correct or failure), it calls itself recursively to pick the next block/condition. Otherwise, it tries to choose the next condition. If there are no more slots available (i.e. maxTrials exceeded), the block is first reset. Once this is controlled for, the next condition (whether incrementally or randomly selected) is picked.

Therefore, the loop breaks when BLOCKnext_cond() is called only to discover that NEXT_BLOCK (i.e. a block is done), or BLOCK_DONE (i.e. for MasterBlock) conditions are met. Thus, if reset_block_circ_buf is set to at_Start or Manual, any repeat attempts to run an already completed block will result in an instant NEXT_BLOCK result, so the block will not be inadvertently repeated.

Appendix B. PCI-DAS1602/12 Information

The PCI-DAS1602/12 board was chosen for support in CORTEX, since it is the one that is most similar to the CIO-DAS16 series ISA board. However, there are some very important differences between these two boards. Before buying this new board, please read the details below to make sure that the board will support your data acquisition needs.

B.1 Technical details:

B.1.1 Fewer digital channels

It is necessary to first give some background about the CIO-DAS16 series boards. There are two parts to the CIO-DAS16 board. One part contains the functionality of a "DAS16" card, including the analog data lines, and 8 digital lines (4 input/4 output). The second part contains the functionality of a "PIO24" card, including 24 digital lines to be used for digital input and output. The 8 digital lines on the "DAS16" portion of the card are used by Cortex as follows:

Digital In #0 - spike 1
Digital In #1 - spike 2
Digital In #2 - bar up/down
Digital In #3 - bar right/left

Digital Out #0 - reset latches
Digital Out #1 - reward
Digital Out #2 - free
Digital Out #3 - free

The PCI-DAS1602 card also contains a "DAS16" part and a "PIO24"part. However, unlike the CIO-DAS16 series, the "DAS16" portion of the PCI-DAS1602/12 card does not contain the extra 4 digital input and 4 digital outputs. Since the PCI-DAS1602 does not contain these extra lines, it was necessary to use the PIO24 portion of the card for these digital lines. Port C was chosen for this purpose, since it can be split as 4 bits input and 4 bits output. It is configured as follows:

Input:
Port C, bit 0 (pin #67) - spike 1
Port C, bit 1 (pin #68) - spike 2
Port C, bit 2 (pin #69) - bar up/down
Port C, bit 3 (pin #70) - bar right/left

Output:
Port C, bit 4 (pin #71) - reset latches
Port C, bit 5 (pin #72) - reward
Port C, bit 6 (pin #73) - free
Port C, bit 7 (pin #74) - free

As with the CIO-DAS16 and the PIO24 boards, Port A is configured as an input port and Port B is configured as an output port.

The NO_INIT feature on the DEVICE line of cortex.cfg will be partially supported for the PCI-DAS1602/12. This is because Port C MUST be initialized as outlined above. If Cortex does not initialize Port C before the start of the timing file, there will be problems with the operation of Cortex. Therefore, the NO_INIT feature can only be used to initialize Ports A & B for the PCI-DAS1602/12, but not for Port C.

B.1.2 Base address differences

For the CIO-DAS16 boards, there is a single base address that is configurable by the user via jumper settings on the board. Once these jumpers are set, this base address value for the board can not be changed without changing the jumpers. Therefore, the user can confidently supply this value in the DEVICE line of the Cortex.cfg file.

The PCI-DAS1602, on the other hand, is a plug-and-play device, which does not have any onboard jumpers to configure the base addresses. Every time the computer boots up, the BIOS assigns base addresses to the board which will not conflict with the other devices in the computer. It is possible that each time that the computer boots up, the board will have different addresses! Also, instead of having just one base address, the PCI-DAS1602 has five base addresses. Because of this complicated situation, Cortex will just query the BIOS and board internally, to figure out these five addresses. The base address that is supplied on the DEVICE PCIDAS1602 line will be ignored, even though it is a required parameter.

B.1.3 Analog data collection differences

For the CIO-DAS16, the analog data is held in two 8-bit registers. The Base+0 register holds the 4 least significant bits, plus the channel number. The Base+1 register holds the 8 most significant bits. For the PCI-DAS1602, however, the analog data register is a 16-bit register, which does **not** contain the channel number. Instead, the code is written to rely on the "fact" (according to ComputerBoards, Inc) that every time a write is issued to the "channel MUX control register", the board will start collecting with the lowest channel number. Therefore, at the beginning of each trial, a write is issued to the MUX channel register. Internal software counters are then implemented, so that the current channel being read can be determined.

Another analog data collection difference is that the PCI-DAS1602 board does not contain a latched bit for the EOC (end-of-conversion) flag. That is, before analog data is read, the EOC bit should be checked to make sure that the analog-to-digital converter is not busy. Since this EOC bit is not latched, it is virtually impossible to check the bit without putting a waiting loop into the code. A waiting loop could easily destroy the timing of spike data collection. According to the technical support representative at Computerboards, Inc., by structuring the code so that the analog data conversion is triggered, followed by the reading of the digital (spike) lines, before reading the analog data, the timing should be allow for accurate data collection.

These differences in the analog data collection should be transparent to the user. They are mentioned here, since the code was written based on word-of-mouth information from the ComputerBoards, Inc. technical support representatives. The official documentation of the board contains minimal information, and does not specify how to overcome these differences.

B.2 Cortex.cfg details:

In order to tell Cortex that you are going to be using the PCI-DAS1602/12 card, you must specify the following lines in your Cortex.cfg file:

DEVICE PCIDAS1602 0x300

As mentioned in the "Fewer Digital Channels" section above, an address must be supplied, but it will be ignored by Cortex. Also, as mentioned above, the INIT and NO_INIT parameter can be used to initialize Port A and B, but not C. (Note: You can not "split out" the PIO24 portion and DASH16 portion of this card and address them separately like can be done with the CIO-DAS16 card.)

Optionally, you can also specify keywords on the DEVICE line to initialize the board to further describe the analog input data. The [<single_or_differential>] parameter allows the user to specify 16 single-ended channels, or 8 differential channels for analog input. It is only used for the PCI-DAS1602/12 board, since it does not have jumpers on the board to set this property. The valid input settings are:

SINGLE - 16 single-ended channels (default, if not supplied)

DIFFERENTIAL - 8 differential channels

Additionally, the user now has the ability to choose whether or not Cortex should send out the reset latch pulse for the PCI-DAS1602/12 only. This would free up another output line that users could access. (IMPORTANT NOTE: This option should only be used by users who are not using Cortex to collect spikes! For example, this would include those who are using a third party spike collection system, or who are not collecting spikes at all.) To specify this option, the user must put the word "NO_LATCH" on the DEVICE line in cortex.cfg. If this parameter is omitted, the reset latch pulse will be sent by default.

The [<bipolar_or_unipolar>] parameter on the DEVICE line, allows the user to specify bipolar or unipolar data setting for analog input. Again, it is only used for PCI-DAS1602/12 board, since it does not have jumpers on the board to set this property. The valid input settings are:

BIPOLAR - bipolar input (default, if not supplied)
UNIPOLAR - unipolar input

NOTE: Cortex expects the analog input to be wired as 16 single-ended channels containing bipolar data. Cortex will initialize the board as such, if the optional DEVICE parameters are not specified, or are incorrectly specified.
If you supply these parameters, you MUST also supply the [initialization] parameter, even though it will be ignored. These 3 parameters must be supplied in the given order, or the default setting (INIT SINGLE BIPOLAR) will be used.

Therefore, the DEVICE line may also look like this, for example:

```
DEVICE    PCIDAS1602        0x300   INIT  SINGLE BIPOLAR
or
DEVICE    PCIDAS1602        0x300   INIT  DIFFERENTIAL UNIPOLAR
```

THREAD PCIDAS1602 defaults 1

A2D_GAIN 1
(Will give values between -5 and +5 volts.)

MULTI_SPIKE PCIDAS1602 0x00 2 1 // 2 spikes from Port A, encoded as 1 and 2
MULTI_SPIKE PCIDAS1602 0x02 2 102 // 2 spikes from Port C, encoded as 102 and 103
etc.

B.3 Hardware details:

B.3.1 Hardware requirements

The following data acquisition hardware is available from ComputerBoards, Inc. (www.computerboards.com):

PCI-DAS1602/12 data acquisition board
C100FF cable
CIO-TERM100 screw terminal board

Note 1: The connector for this board requires 100 pins. Therefore, the cabling and interface boxes that you have from the CIO-DAS16 boards will not work, since they are 37-pin connectors.

Note 2: The spike channels must still be latched. Therefore, you must either alter the CIO-TERM100 screw terminal board to contain the latches, or have another device that performs the latching. For testing, I use a device built at NIH (RSB #8668) which can latch up to 8 spike channels. The circuit diagram for this device can be found at http://www.cortex.salk.edu/CortexCircuit.php.

B.3.2 Important Pin Numbers

Pin #	Function
4	Analog input Ch 1 High (EPP_x)
6	Analog input Ch 2 High (EPP_y)
8	Analog input Ch 3 High (EOG_x)
10	Analog input Ch 4 High (EOG_y)
1 and 18	Analog GNDs
51 through 58	Port A Digital Input
59 through 66	Port B Digital Output
67	Port C bit 0 - spike input
68	Port C bit 1 - spike input
69	Port C bit 2 - bar up/down
70	Port C bit 3 - bar left/right
71	Port C bit 4 - reset latches
72	Port C bit 5 - reward
73	free (output bit)
74	free (output bit)
89, 91, 100	GNDs

B.4 Using DEVinp() and DEVoutp():

The DEVinp() and DEVoutp() functions can be used in a timing file to read and write a byte of data to a specified port number. In the PCI-DAS1602/12 manual, Port A occupies register BADR3+4, PortB occupies BADR3+5, and Port C occupies BADR3+6. Since this is pretty confusing to remember, these port numbers are remapped internally in Cortex, so the more intuitive port numbers can be used:

```
PortA =  0
PortB =  1
PortC =  2
```

For example, assuming that the only device in use is the PCI-DAS1602 board, the code in the timing file could be as follows:

```
// read a byte from Port A
val = DEVinp(0, 0x0);

// manually write a byte containing all 1's to Port B
DEVoutp(0, 0x1, 0xff);
```

Appendix C: Sample Cortex.cfg file

The following is an example of the Cortex.cfg file that is used to customize Cortex. Double slashes at the front of the line indicates a comment and are not interpreted.

```
// ************************* CORTEX.CFG ***********************
// This file gives an adequate setup
// N.B. There are many entries which might appear confusing.  In general,
// there are defaults for just about everything, so an empty cortex.cfg file
// will still allow CORTEX to run, but in a very pared down way.
// Note that most key words (esp. the first one on a line) are not case
// sensitive.
// For those who wonder why so many defaults are specified here:  CORTEX
// is getting so large that its default configuration doesn't run on some
// systems with limited memory.  Therefore, the cortex.cfg file lets you
// remove features you don't use, thus freeing needed memory.  This is
// especially true of the DATA_STRUCTS line

// TEXT_COLORS: <normal> <bold> <highlighted> <blinking>
//   the numbers below specify adequate defaults for Hercules or VGA

TEXT_COLORS       0x07    0x0f    0x70    0x87 25

// MONITOR_TYPE:  <monitor_type> <font_path> <font_choice>
//   monitor_type:  DEFAULT, NONE, MONOCHROME, HERCULES, CGA, EGA, VGA
//   font_path: any of the .fon files located in the /fonts directory
//     font_path can now just be the name of a file located in the
//     CORTDIR directory -- JAM 08-May-96
//   font_choice:  the syntax is from Microsoft C 7.0.  Example should suffice
//        other choices include    courb.fon "t'courier'h9w5b"  and
//        tmsrb.fon "t'tms rmn'h9w5b'  , for example

//MONITOR_TYPE        DEFAULT helvb.fon  "t'helv'h9w5b"
MONITOR_TYPE      VGA helvb.fon  "t'helv'h9w5b"

// GRAPHICS_SPECS: <Xdim> <Ydim> <frames_per_sec> <pixels_per_dva>
//    <bits_per_pixel> <re-init_on_depth_conflict> [<background_constant>]
//    Xdim & Ydim:  pixel dimensions of subject's screen
//    frames_per_sec:  refresh rate of subject's screen
//    pixels_per_dva:  number of pixels per subject's degree of visual angle
//       bits_per_pixel: for subject: e.g. 1,2,4,8,16,24,32
//    re-init_on_depth_conflict:   allows reset when a new bitmap's depth
//        differs from that to which the subject's screen was initialized.
//        I don't think that anyone uses this feature.
//  background_constant:  an optional parameter to allow the user to
//  specify whether or not they want the background color to remain constant
//  throughout the execution of Cortex.  If they omit the parameter, the
//  screen will go black at the end of the trials, instead of keeping the
//  same color as during the trial.   The choices are:  CONSTANT or BLACK.
//

//GRAPHICS_SPECS      640 480 72   34.5    34.5    8   NEVER
//GRAPHICS_SPECS      640 480 60 35 35 8   NEVER  CONSTANT
GRAPHICS_SPECS        640 480 60 35 35 8   NEVER

// SOUND: <speaker_config> <speaker_geom>
```

```
// speaker_config: type of speakers that will be used
//      HEADPHONE, MONO, QUAD, STEREO, SURROUND, or DEFAULT (stereo)
// speaker_geom: used to specify the angle of the two stereo
//      speakers from each other.  Valid values are:
//  MIN = 5 degrees
//  NARROW = 10 degrees
//  WIDE = 20 degrees (default)
//  MAX = 180 degrees
//  DEFAULT  (wide)
// The geometry values are only used if the speaker configuration
// is STEREO.  (Since two parameters are required on the SOUND line,
// just put DEFAULT as the speaker_geom parameter when using
// configuration settings other than STEREO.)
//
// Sound is only available in the two computer 32-bit version of Cortex.
// Additionally, the SOUND configuration values are only used by
// the DirectX version of receive, and are ignored by the DOS version.
//
SOUND  STEREO  WIDE


// PLAY <#play_items> <float max_x> <float max_y> (in dva)
// This line is NOT necessary for most people.  In NNIOS, the PLAY item has
// a completely separate bitmap, so you must define ahead of time how large
// it can be.  In TIGA, this isn't as necessary.

PLAY     1   7.5 7.5

// THREAD_MANAGER <max_#interrupts> <max_#threads>
// This is only needed by people adding many CSS threads or writing their
// own interrupt and thread routines.  The defaults (5,10), are about three
// times as many as are normally used, so the defaults should suffice for
// everyone for quite a while.  CORTEX will warn you if there aren't enough.

//THREAD_MANAGER      7   10

//** USER SCREEN MAPPING **
// The lower left hand coordinate of the screen is (0,0), and the upper
// right hand corner is (10000,10000) in this mapping.  These integer values
// will be mapped to the actual size of the user screen.  For example, VGA
// uses 640x480, but Hercules uses 720x348.  By using 0->10000 instead of
// pixel coordinates, one cortex.cfg file can be used for all monitor
// configurations.  Lower Left is abbreviated <ll>, and Upper Right is <ur>.

// HISTOGRAM <htg#> <ll.x> <ll.y> <ur.x> <ur.y>
// There can be up to <#spike_channels> histograms shown on the screen.  Each
// Histogram automatically includes areas for this histogram, raster_table,
// and raster line.  The raster line dynamically shows the spike inputs.  The
// raster_table shows the past history of spike inputs.  As usual, the
// <ll.x,y> and <ur.x,y> are in User Screen Mapping coordinates
// Remember that the lower left hand corner is (0,0)

//Histogram           1   0   5200    3000    9900
//Histogram           2   0   0   3000    4700

Histogram         1   0   0   3000    2500
Histogram         2   0   2500    3000    5000
Histogram         3   0   5000    3000    7500
Histogram         4   0   7500    3000    10000
```

```
// STATUS_RECT    <ll.x> <ll.y> <ur.x> <ur.y> <#columns>
// Status_rect defines the region of the user screen in which messages are
// printed from the GMENU system (graphical menu).  Cortex automatically
// formats all of the GMENU entries in the order requested so that they look
// pretty.  The only thing Cortex needs to know is how many columns you want.
// The number of entries does not have to be an even divisor of #columns: so
// if you have 10 entries and specify 3 columns, they will be 4:4:2

STATUS_RECT       3250        8025    10000   10000   3

// GMENU GETS<something> <label_string> <value_length>
// GMENU allows you to pick and choose not only what is shown on the user
// screen, but also determine where it will be shown, and how much space it
// will have to show its message.  All of the available functions are shown
// here, although some of them are commented out.  Since three columns have
// been specified from STATUS_RECT, the GMENU entries have been divided into
// three groups for visual convenience.  This also makes it easier to
// ensure that they are lined up nicely.  For example, in the first block,
// "GetData" is the longest entry, so an extra space or two are added to the
// others so that the values returned by those functions all start aligned.
// On the other hand, the GETSmessage? entries do not specify a label string
// so that they can use the entire space for their messages.

GMENU GETScycle_num        "cycle# "    4
GMENU GETSblock_num        "block# "    4
GMENU GETScond_num     "cond#  "    4
GMENU GETStrial_num        "trial# "    4
GMENU GETShtg_scale        "scale  "    4
GMENU GETSbin_width        "bwidth "    4
GMENU GETScollect_data "GetData"    4

GMENU GETSsaccade          "saccade"    10
//GMENU   GETSdynamic_fixwin_size  "fixwin "    10
GMENU GETSwindowX          "fixwinX"    10
GMENU GETSwindowY          "fixwinY"    10
GMENU GETScond_pct_correct "Cpct"  15
GMENU GETScond_circ_buf        "Cbuf"  15
GMENU GETSblock_pct_correct    "Bpct"  15
GMENU GETSblock_circ_buf       "Bbuf"  15

GMENU GETSmessage1     ""      50
GMENU GETSmessage2     ""      50
GMENU GETSmessage3     ""      50
GMENU GETSmessage4     ""      50
GMENU GETSmessage5     ""      50
GMENU GETSfile_name        "file  "25
GMENU GETStrial_status "status"25

//GMENU   GETSnothing  ""  1

// EOG_MAPPING <minV> <maxV> <ManualDVA?> <.5 Xdva> <.5 Ydva>
//    <ll.x> <ll.y> <ur.x> <ur.y> <#eyewins>
// minV and maxV: the minimum and maximum values returned by the a2d board.
//    for most people, a 12 bit analog2digital converter is used, so the
//    values range from 0->4096.  For right now, CORTEX converts these to
//    -2048->+2048, but that is not crucial.  The minV and maxV are not fully
//    implemented pending a decision on how to deal with this issue, so leave
//    them alone.
```

```
// ManualDVA?: {0,1,2}
//    (0) minV/maxV will map to the screen size, as determined from
//        <Xdim> <Ydim> and <pixels_per_dva>. Adjust your gains accordingly.
//    (1) <Xdva> and <Ydva> (twice values entered) will map to minV/maxV.
//        The screen size is read, and pixels_per_dva remains correct.  The
//        eog_window on the user's screen reflects the size of Xdva and Ydva,
//        not the screen size.
//    (2) minV/maxV will map to width of the screen size.  Exactly like option
//        0, except maps to (x,x) instead of (x,y).  Means gains equal?
// <ll.xy><ur.xy>: Defines the location of the eog_window on the user screen
//    in User Screen Coordinates.
// <#eyewins>: CORTEX keeps track of the location of every object drawn to
//    the screen.  There are up to 10 test_screens per condition (TEST0->
//    TEST9), each test_screen can have up to 16 objects, and each test_screen
//    can be moved around on the screen by SWEEPwin() or other commands.
//    CORTEX keeps track of all of the objects' positions dynamically so that
//    you can ask ITEM_POSeye_is_here() or other commands to determine whether
//    the subject is looking at a particular object.  In addition, the user
//    may want to keep track of positions on the screen at which there are
//    no objects present, such as the positions of objects from the previous
//    condition.  In order to do this, you need to be able to store and
//    access/query other item positions.  This is what the eye_wins are for.
//    Before you run CORTEX, you must guess how many you will need (default is
//    15).  Once in CORTEX, you can set these values manually or using CSS.

EOG_MAPPING       -2048 2048 1 10 10 3250 0 10000 8000 10

// EOG_STYLE is new; it's an integer 0-2:
//   0 is regular NIH style for EOG (it's the default)
//   1 is like NIH, but dots are not erased automatically, you must
//     use the new clear_eog() function in your CSS file to clear
//      things at the start of each trial.
//   2 is lines connecting the points, also requires CSS file to
//     call clear_eog() periodically..
//      -- JAM 30-Apr-96
EOG_STYLE 0

// DEVICE <char *device> <int addr>    [<initialization>]
// [<single_or_differential>] [<bipolar_or_unipolar>]
// device: string of METRABYTE, PIO24, COMPUBOARD, PIO96, NO_DEVICE,
//    RANDOM_SPIKE_DEVICE, DASH16, PCIDAS1602
// addr: The base port for the board.  Examples are listed below
//
// [initialization]:  an optional parameter to determine whether or not to
// allow Cortex to send an initialization byte to the board.
// Valid choices are:  INIT (Cortex will initialize the board)
// or NO_INIT (user must initialize the board).  If this parameter is
// omitted, the default behavior is for Cortex to initialize the board.
// See more details on the Cortex web page.
//
// [<single_or_differential>]: Allows the user to specify 16 single-ended
//   channels, or 8 differential channels for analog input.  Only used
//   for PCI-DAS1602/12 board, since it does not have jumpers on the board
//   to set this property.  If this parameter is supplied for the other
//   types of boards, it will be ignored.
//   Valid input settings:
//       SINGLE -  16 single-ended channels  (default, if not supplied)
//       DIFFERENTIAL - 8 differential channels
//   Please note the following:
```

```
//     1) Cortex expects the analog input to be wired as 16 single-ended
//          channels.
//     2) If you supply this parameters, you MUST also supply the [initialization]
//          parameter and the [bipolar_or_unipolar] parameter.  These parameters
//          must be supplied in the given order, or the default setting
//          (INIT SINGLE BIPOLAR) will be used.
//
// [<bipolar_or_unipolar>]: Allows the user to specify bipolar or
//   unipolar data setting for analog input.  Only used
//   for PCI-DAS1602/12 board, since it does not have jumpers on the board
//   to set this property.  If this parameter is supplied for the other
//   types of boards, it will be ignored.
//   Valid input settings:
//       BIPOLAR -  bipolar input (default, if not supplied)
//       UNIPOLAR - unipolar input
//   Please note the following:
//     1) Cortex expects the analog input to be bipolar input.
//     2) If you supply this parameters, you MUST also supply the [initialization]
//          parameter and the [single_or_differential] parameter.  These parameters
//          must be supplied in the given order, or the default setting
//          (INIT SINGLE BIPOLAR) will be used.
//
// CORTEX now uses threads in its interrupt routines in an effort to make
// it easier to add new data aquisition boards and/or interface with other
// devices.  Use the DEVICE line to declare which boards you have in your
// system and want to use.  If you do not declare a DEVICE, you can not
// use it in a THREAD statement.
// The value returned from DEVICE (base 0) is the value needed if you want
// to use the DEVinp() and DEVoutp() functions from CSS.
// Of couse, you can also just call outp() and inp() directly if you know
// the addresses.

DEVICE        COMPUBOARD           0x300

//DEVICE      COMPUBOARD           0x300   INIT
//DEVICE      COMPUBOARD           0x300   NO_INIT
//DEVICE      PCIDAS1602      0x300
//DEVICE      PCIDAS1602      0x300   INIT  SINGLE BIPOLAR
//DEVICE      PCIDAS1602      0x300   INIT  DIFFERENTIAL UNIPOLAR
//DEVICE      random_spike_device  0
//DEVICE      DASH16               0x300
//DEVICE      METRABYTE            0x300
//DEVICE      PIO24                0x310
//DEVICE      COMPUBOARD           0x300
//DEVICE      PIO96                0x310
//DEVICE      no_device            0
//DEVICE      PCI_DIO24            0x300

// THREAD <char *device> <char *thread> <int ms_per_tik>
// <char *device>: same as above {METRABYTE, COMPUBOARD, ...}
// <int ms_per_tik>:  How many milliseconds between running of thread.
// <char *thread>: all of the allowed <thread> names are used below
//    defaults:  Should be run at 1000Hz (ms_per_tik = 1).
//        This special function/interrupt does everything that
//        is done by read_lines, SPIKEstoreAll, read_eog and EOGstore, all
//        in one function, thus being faster because of decreased overhead.
//        It is recommended that users use this function instead of the
//        other four unless there is some pressing need to do otherwise.
//        If the bar_state and reward threads are not specifically specified,
```

```
//        then CORTEX will use the same board for these as for the default
//        threads.  bar_state and reward will individually override this
//        default.  However, if defaults and any of the other threads are
//        both specified, both will be run.
//
//    ** Less commonly used functions **
//    bar_state: not a thread (so ms_per_tik = 0).  Tell CORTEX which board
//        to use to read the bar_state from get_bar_state() in CSS
//    reward: not a thread (so ms_per_tik = 0).  Tell CORTEX which "board" to
//        issue the reward() from CSS
//    read_lines: Should be run at 1000Hz (ms_per_tik = 1).  Reads the spike
//        data from the specified board into the int DevControl.inputLines
//    read_eog: Should be run at 1000Hz (ms_per_tik = 1).  Reads the eye
//        position using the specified board.
//    SPIKEstoreALL: Should be run at 1000Hz, and MUST follow read_lines.
//        Assumes that DevControl.inputLines has been updated.  Checks the
//        first two bits (spike channels) and stores spike codes if occured.
//
//    ** Threads whose rates are controlled from CORTEX menus, not here **
//    RASTERupdate: is a thread, whose rate is determined by the bin size
//        of the on-line histograms.  CORTEX uses the value passed to
//        start_trial() to initialize the RASTERupdate() thread.  However,
//        you can tell CORTEX not to run that thread by specifying a negative
//        number for the bin size.  The histograms will still be computed
//        and stored correctly, but the raster line will not be drawn.
//    EOGupdate: specifies the rate at which the eyespot on the user screen
//        is refreshed.  A good rate is 15Hz (every 66ms).  If done too fast,
//        you can't see it.  It is set from the Run:Params:General menu, and
//        the default is 66ms.
//    EOGstore: is an interrupt whose rate is set in the Run:Params:General
//        menu, not here (the value you give it is ignored).  The initial
//        value is 0, so if you want to store eog data, you must set it to
//        a positive number, as well as call put_eye_data_in_buf() from CSS.
//
// You need to specify which threads will be run from the interrupt,
// and at what rate they should be run.  Right now, the user has more
// control than s/he should, so the choices may be restricted in the near
// future.  If you wonder why non-threads are specified, it is because
// different boards might be used to do that operation.

THREAD    COMPUBOARD              defaults    1
//THREAD  PCIDAS1602              defaults    1
//THREAD  PCIDIO24            defaults    1
//THREAD  RANDOM_SPIKE_DEVICE     defaults    1

//THREAD  METRABYTE               bar_state       0
//THREAD  METRABYTE               reward          0
//THREAD  RANDOM_SPIKE_DEVICE     read_lines      1
//THREAD  RANDOM_SPIKE_DEVICE     SPIKEstoreAll   1
//THREAD  METRABYTE               read_eog    1
//THREAD  DASH16                  chain_INT08     18

// DATA_STRUCTS <#CODE_ISI_els> <#EPP_els> <#EOG_els> [all as max per trial]
// <#CODE_ISIels>:  The max number of encode (CODE and timestamp) elements
//   which can be stored per trial.
// <#EPPels>: The max number of EPP (can really be anything: sizeof(char))
//   which can be stored per trial.
// <#EOGels>: The max number of eye positions which can be stored
//   per trial
```

```
// The defaults for all of these used to be 4000:0:2000, and are the defaults
// here as well.  However, if you know that you will never use EPPbuf, then
// why bother allocating space you don't need.  OR, if you want to store
// more extended EOG tracings, you might need more room.

DATA_STRUCTS 20000 5000 20000

// A2D_CHANNELS is normally 2.  If you need to cycle between 4 A2D channels,
// storing the data from channels 1 and 2 into the EPP buffer, and channels
// 3 and 4 into the EOG buffer, than set A2D_CHANNELS to 4

//A2D_CHANNELS    4

// A2D_GAIN is specific for the METRABYTE/COMPUBOARD a/d device. These
// boards have a variable gain stage that can be set under software
// control to 4 possible values (the exact gains depend on the particular
// model of the board -- see your manual for details). The config
// option should be set to -1, 0, 1, 2 or 3.  -1 indicates use the
// hardware default.
//      -- JAM 24-Apr-96

A2D_GAIN  1

// TOUCH_SCREEN   <int ms_per_tik> [<store_touch_data>] [<invert_y_data>]
// To use a touch screen, uncomment this TOUCH_SCREEN line below
// To emulate the touch screen using the mouse, use a negative number for
// the ms_per_tik value.
// TMN 01/24/00 Added the optional parameter <store_touch_data> to
// allow the user to specify whether or not the touch values should
// be stored to the EPPbuf, the CODE/ISI buf, both places, or neither
// place.   Valid values are:
//      EPPBUF
//      CODEBUF
//      NEITHER
//      BOTH  (default)
// If no value is provided, the touch data will be stored to both the
// EPPbuf and CODE/ISI buf.
// [<invert_y_data>]:  an optional parameter to allow the user to specify
// whether or not the Y value from the touchscreen should be inverted.
// If the user does not specify this parameter, the default behavior is
// to *not* invert the y.  Note:  This change has only been carried out
// for the two-computer version of Cortex.  For the other versions,
// this parameter is not used.   Valid values are:
//       INVERT_Y
//       NO_INVERT
//TOUCH_SCREEN    -50
//TOUCH_SCREEN -50  BOTH
//TOUCH_SCREEN -50  BOTH  NO_INVERT
//TOUCH_SCREEN -50  BOTH  INVERT_Y


// LUT    <int num_palettes>
// This command doubles for the LUT:Number menu within CORTEX, allowing one
// to specify the number of palettes to create when launching CORTEX
LUT    0

COM_PORT              1       115200
//COM_PORT        1   19200
```

```
////////////////////////////////////////////////////////////////////////
// MULTI_SPIKE    <char *device> <port_offset> <#spikes> <starting_encode_val>
//      [<starting_bit>]
////////////////////////////////////////////////////////////////////////
// This syntax will hopefully replace the SPIKE_CHANNELS and much of the
// THREADS defaults stuff.  For the sake of backwards compatibility,
// the other syntaxes will remain, although they will call these functions
// internally.  However, if MULTI_SPIKE is specified at all, it will
// supercede anything entered for SPIKE_CHANNELS or THREAD defaults.
//
// CORTEX will ensure that invalid ports are not specified for known devices,
// but will not check for overlap of spike values, or multiple reads from
// a given device's port.
// TMW 10/21/96
//
// Please refer to the documentation that was shipped with the data
// acquisition board when specifying the base address and port offset.
// For example, the port offsets for the Compuboard CIO-AD16 are:
// BASE+0x10(A), BASE+0x11(B), and BASE+0x12(C).   The port offsets for
// the Compuboard CIO-DAS1600 series boards are: BASE+0x400(A),
// BASE+0x401(B), and BASE+0x402(C).
// TMN 02/11/99
//
// The optional parameter <starting_bit> gives the user more flexibility in
// specifying which bits correspond to which spikes and encode values.  For
// example, if one wants to use the DASH16 to collect 4 spike channels, using
// the predefined spike encodes, one might use the following commands, where
// SPIKE1 = 1, and SPIKE3 = 102:
//     MULTI_SPIKE DASH16 0x03 2 SPIKE1 0
//     MULTI_SPIKE DASH16 0x03 2 SPIKE3 2
// TMW 11/10/96
////////////////////////////////////////////////////////////////////////

MULTI_SPIKE   COMPUBOARD    0x03  2   1   // default for DASH16

//MULTI_SPIKE   COMPUBOARD  0x03   2   102 2   // spikes 3&4 from DASH16
//MULTI_SPIKE   COMPUBOARD  0x10  8   104  // 8 spikes from PIO24 port A
//MULTI_SPIKE   COMPUBOARD  0x12   4   112 // 4 spikes from PIO24 port C
//MULTI_SPIKE   PCIDAS1602  0x00  2   1   // 2 spikes from Port A of PCI-DAS1602
//MULTI_SPIKE   PCIDAS1602  0x02  2   102 // 2 spikes from Port C of PCI-DAS1602
//MULTI_SPIKE   PCIDIO24    0x00  2   1   // 2 spikes from Port A of PCI-DIO24
//MULTI_SPIKE   PCIDIO24    0x02  2   102 // 2 spikes from Port C of PCI-DIO24
//MULTI_SPIKE   DASH16  0x03  2   1   // default for DASH16
//MULTI_SPIKE   DASH16  0x03  2   102  2  // spikes 3&4 from DASH16
//MULTI_SPIKE   PIO24       0x02  4   102 // 4 spikes from PIO24 port C
//MULTI_SPIKE   PIO24       0x00  8   1004    // 8 spikes from PIO24 port A

// SPIKE_CHANNELS <#spike_channels>
// Obsolete:  If SPIKE_CHANNELS is used, it will map to
// MULTI_SPIKE    DASH16  0x03    <#spike_channels>    1
// TMW:  10/21/96

//SPIKE_CHANNELS      2
```

Copyright notice

Conditions for Using Cortex

Some portions of Cortex (and associated utility programs, including Procortex, STATS100, and GRAST) are copyrighted (see History of Cortex section for a list of its authors). The program is "freeware". That is, the program and its source code are provided free of charge and may be copied freely, provided that this notice is attached. However, its use is subject to certain restrictions, and, thus, it is not strictly speaking in the public domain. The primary restriction is that you must not sell Cortex or its utility programs or incorporate it into any program that is sold. Furthermore, the Cortex program itself includes code from the Bison compiler generator, which is copyrighted by the Free Software Foundation (FSF). The FSF requires as part of the license for Bison that no program incorporating Bison code be sold. The Bison license is included in the source distribution for Cortex, along with complete information on obtaining the source code for Bison.

The source code for Cortex is provided freely on request, and we encourage users to modify and add to it. Depending on the version of the code you receive, one or two object modules required for linking the source code may be commercial products. If you plan to work with the source code, you should purchase any necessary commercial products directly from the appropriate vendor. Hopefully, all commercial object modules will have been eliminated by the time you receive the code, at least for the Cortex program itself. If you improve the source code, add to it, or write any supplemental programs for creating stimuli or analyzing data from Cortex, we ask to be sent copies. If we distribute your code to new users, you will be acknowledged. The program is provided "as is", and we accept no liability whatsoever for damages or loss of any kind caused by its use. Use of the program implies acceptance of all of these conditions.