# Optimizing Remote Measurement Speed for the Agilent 8614xB Series of Optical Spectrum Analyzers

Product Note 86140-3

Agilent Technologies

It has long been known that using automated test equipment (ATE) is a great way to speed up testing times and thereby reduce overall manufacturing time, increase production volume, and reduce cost of test. Firmware versions B.04.00 and later of the Agilent 8614xB series of Optical Spectrum Analyzers (OSA) contain two features that increase ATE speed even more. First, the front panel display of the OSA can now be turned off during remote operation. This feature provides a significant improvement in the measurement speed of the instrument, as processor power is no longer used to update the display. Second, a GPIB command buffer can be enabled so that the OSA will behave like most other GPIB instruments and accept several commands in quick succession. This feature will remain disabled by default so that programs written for firmware version prior to B.04.00 will be fully compatible with the newer versions of the OSA firmware. With these improvements, overall program execution times can be reduced on the order of 30 to 50 percent. Individual results will vary, however, due to such factors as application, the controller and GPIB hardware, and specific commands used.

## "Display-off" Operation Mode

Constantly updating the OSA display uses up a significant amount of computing power and slows down the instrument. Changing almost any setting or running any operation changes the instrument display. If the OSA display is turned off, this step is eliminated and measurement speed is greatly increased. In OSA firmware versions B.04.00 and later, a single command, *DISPlay[:WINDow[1]] OFF*, can turn off the display and greatly increase the overall speed of the instrument in almost all remote operations. The display can easily be re-enabled by sending the inverse command, *DISPlay[:WINDow[1]] ON*, or by pressing the front panel *Local* button. The process of switching the display on or off usually requires between 10 and 15 seconds, but this is easily recouped in the time saved from disabling the display.

Several common processes were simulated and tested to measure the time saved by turning off the display. A description of each of these tests can be found in Appendix A, the test program source code can be found in Appendix B, and a full description of the test set up and equipment used can be found in Appendix C. Table 1 lists the average of 10 test times with the display both enabled and disabled for each of these processes. It also lists the absolute and percentage time saved for each process. The percentage time saved is calculated by dividing the absolute time saved by the test time with the display enabled. Notice that these results have a very low standard deviation meaning that they are highly repeatable.

*Table 1. Test Statistics*

| Test | Display Setting | Average Test Time (ms) | Standard Deviation (ms) | Absolute Time Savings (ms) | Percent Time Savings |
|---|---|---|---|---|---|
| **Reset** | ON | 3875.3 | 117.3 | 2046.7 | 52.81% |
| | OFF | 1828.6 | 21.1 | | |
| **AutoMeasure** | ON | 14005.2 | 195.0 | 5268.8 | 37.62% |
| | OFF | 8736.4 | 153.7 | | |
| **AutoAlign** | ON | 28340.8 | 168.5 | 9189.3 | 32.42% |
| | OFF | 19151.5 | 65.9 | | |
| **Zoom** | ON | 5006.2 | 112.8 | 1908.1 | 38.11% |
| | OFF | 3098.1 | 113.9 | | |
| **Bandwidth** | ON | 9282.4 | 148.5 | 3882.4 | 41.83% |
| | OFF | 5400 | 104.6 | | |
| **Markers** | ON | 4466.5 | 225.5 | 2250.4 | 50.38% |
| | OFF | 2216.1 | 121.5 | | |
| **Integration** | ON | 7213.3 | 188.6 | 3462.0 | 47.99% |
| | OFF | 3751.3 | 23.2 | | |
| **SMSR** | ON | 9340.3 | 134.4 | 4082.1 | 43.70% |
| | OFF | 5258.2 | 168.6 | | |
| **OSNR** | ON | 19910.5 | 193.9 | 9188.0 | 46.15% |
| | OFF | 10722.5 | 161.5 | | |
| **Trace Download** | ON | 1070.5 | 116.1 | 512.8 | 47.90% |
| | OFF | 557.7 | 12.7 | | |
| **Function** | ON | 6119.7 | 161.1 | 3418.8 | 55.87% |
| | OFF | 2700.9 | 38.0 | | |

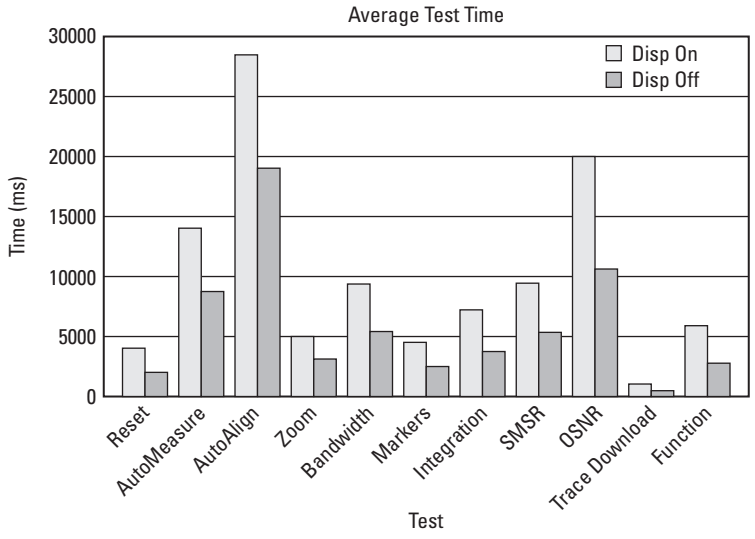


*Figure 1. Program Execution Times with Display On and Off*

## GPIB Command Buffer

Versions of the 8614xB OSA firmware before B.04.00 allow only one command to be sent to the OSA at a time. If a second command is sent before the first is finished, the GPIB bus will simply hang until the first command is finished at which point the second command will be read by the instrument. The reason behind this is that GPIB relies on a three-wire handshaking system between the controller and the instrument to ensure proper communications. The OSA keeps one of these control lines, the *NRFD* (not ready for data) line, high until it finishes with each command. This means that the controller is unable to send any more commands until the OSA is finished with that command and the controller will be unable to send multiple commands in quick succession. This also prevents the controller from communicating with any other instruments on the bus while the OSA is processing a command.

The advantage is that there is no need for program synchronization because commands cannot be executed out of sequence, as only one command is processed in the OSA at any given time. The disadvantage to this approach is that overall program speeds are decreased as the controller is held up as the OSA processes each command.

The command buffer in the 8614xB firmware after version B.04.00 allows the instrument to receive several commands in quick succession without having to worry about tying up the bus. Each command is placed in the buffer as it comes in and the NRFD bit remains low. Figure 2 illustrates this process. For example, if a high-resolution sweep is being performed, the commands that perform the data calculations can be sent before the sweep is completed. The disadvantage is that the program now requires synchronization to ensure that operations occur sequentially. Again, synchronization is only required with the buffer. In the example above, the data calculations may be attempted before the sweep is completed, but they will not be performed correctly.

Synchronization can be accomplished by several different methods. The simplest is to use the **OPC?* (operation complete) query. This query will return a "1" when the most recent operation is complete. If the controller is set up to wait for this response, it will not send the next command until the instrument has completed all of its previous tasks. Another simple command is **WAI* (wait). If this command is sent to the OSA, it will wait until all of the present tasks are completed before continuing on to the next command. This eliminates the need for the controller to wait for any response from the instrument.

The command buffer is enabled using the command *SYSTem:COMMunication:GPIB:BUFFer ON*. Similarly, it is disabled with the command *SYSTem:COMMunication:GPIB:BUFFer OFF*. With the buffer disabled any existing 8614xB code will perform exactly as it did with the versions of the firmware B.03.01 and earlier. The command buffer is disabled by default so it must be enabled at the beginning of any program in which it is utilized.

**Buffer Disabled**
**(and OSA firmware before B.03.01)**

While the first command is being processed by the OSA, the NRFD bit is set high, which prevents any further commands from being sent over the bus.

**Buffer Enabled**

With the command buffer, other commands are free to move to the OSA and other instruments on the bus even while the OSA is processing the first command.

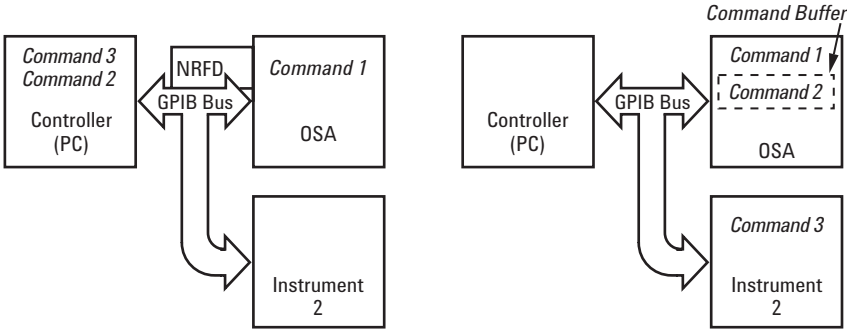


***Figure 2. OSA Command Buffer Modes***

This firmware upgrade is free to any OSA owner and can be downloaded or ordered from www.agilent.com or contact your local sales office for more details.

For more information refer to the following:
8614xB User's and Programmer's Guide (part number 86140-9000)

# Appendix A - Test Descriptions

## Test 1 - Reset

| | | |
|---|---|---|
| **Command Used** | *RST | Resets the instrument |
| **Average Test Time (Display On) - ms** | 3875.3 | |
| **Average Test Time (Display Off) - ms** | 1828.6 | |
| **Percent Time Saving** | 52.81% | |

This test was a simple one-command instrument preset. The only command used is the IEEE 488.2 required command **RST*.

## Test 2 - AutoMeasure

| | | |
|---|---|---|
| **Command Used** | DISP:WIND:TRAC:ALL:SCAL:AUTO | AutoMeasure |
| **Average Test Time (Display On) - ms** | 14005.2 | |
| **Average Test Time (Display Off) - ms** | 8736.4 | |
| **Percent Time Saving** | 37.62% | |

The AutoMeasure command, the only one used in this test, automatically scales the instrument display to encompass the largest input source and places a marker at the peak power. This command actually changes many settings and performs multiple operations, so despite the fact that AutoMeasure is designed mainly for front panel operation, it serves as a good benchmark of general instrument use.

## Test 3 - AutoAlign

| | | |
|---|---|---|
| **Command Used** | CAL:ALIG:AUTO | AutoAlign |
| **Average Test Time (Display On) - ms** | 28340.8 | |
| **Average Test Time (Display Off) - ms** | 19151.5 | |
| **Percent Time Saving** | 32.42% | |

AutoAlign aligns the monochromator output with the photodetector at the wavelength with the highest power. This is an important procedure to perform each time that the OSA has been moved, subject to large temperature change, or after warm up. Even though this procedure deals primarily with the optics of the instrument, there is still a significant time savings from disabling the display. The time savings from a single AutoAlign almost offsets the time used to disable the display.

## Test 4 - Zoom

| | | |
|---|---|---|
| **Commands Used** | INIT:IMM | Run a single sweep |
| | CALC1:MARK1:MAX | Marker to peak power |
| | CALC1:MARK1:SCEN | Marker to center |
| | SENS:WAV:SPAN 10nm | Set the wavelength span |
| | CALC1:MARK1:X? | Get marker wavelength |
| | CALC2:MARK1:Y? | Get marker amplitude |
| **Average Test Time (Display On) - ms** | 5006.2 | |
| **Average Test Time (Display Off) - ms** | 3098.1 | |
| **Percent Time Saving** | 38.11% | |

The zoom test finds the peak power, zooms in on it, runs another sweep on that small area, and finally retrieves the coordinates of that point. This test is based on Example 2 in the 8614xB User's manual. This test updates the display several times so there is still significant improvement realized by disabling the display.

## Test 5 - Bandwidth Measurement

| | | |
|---|---|---|
| **Commands Used** | SENS:WAV:STAR 1530NM | Set start wavelength |
| | SENS:WAV:STOP 1570NM | Set stop wavelength |
| | SENS:POW:DC:RANG:LOW –60DBM | Set sensitivity |
| | INIT:IMM | Take sweep |
| | CALC1:MARK1:MAX | Marker to peak |
| | CALC1:MARK1:SCEN | Marker to center |
| | CALC1:MARK1:X? | Read marker wavelength |
| | CALC1:MARK1:Y? | Read marker amplitude |
| | SENS:BWID:RES 0.1 NM | Set resolution bandwidth |
| | SENS:WAV:SPAN 2NM | Set span |
| | CALC1:MARK1:FUNC:BWID:NDB –20.0DB | Select dB down where BW is calculated |
| | CALC1:MARK1:FUNC:BWID:INT ON | Enable BW marker interpolation |
| | CALC1:MARK1:FUNC:BWID:READ WAV | Sets the BW unit of measurement to WL |
| | CALC1:MARK1:FUNC:BWID:STAT ON | Enable bandwidth marker |
| | CALC1:MARK1:FUNC:BWID:RES? | Returns axis values between markers |
| **Average Test Time (Display On) - ms** | 9282.4 | |
| **Average Test Time (Display Off) - ms** | 5400.0 | |
| **Percent Time Saving** | 41.83% | |

The bandwidth test measures the bandwidth of the input source. This test is based on Example 3 from the 8614xB User's manual and builds on Test 4. It essentially focuses in on the peak power and then repeats the sweep at a very fine resolution bandwidth to increase the accuracy of the bandwidth measurement. The small amount of math involved in calculating the bandwidth is responsible for the slight increase in the amount of time saved over Test 4.

## Test 6 - Markers

| | | |
|---|---|---|
| **Commands Used** | SENS:WAV:STAR 1480nm | Set start wavelength |
| | SENS:WAV:STOP 1580nm | Set stop wavelength |
| | INIT:IMM | Run a single sweep |
| | CALC:MARK:X:WAV 1480nm | Sets the marker position |
| | CALC:MARK:Y? | Reads the marker amplitude |
| **Average Test Time (Display On) - ms** | 4466.5 | |
| **Average Test Time (Display Off) - ms** | 2216.1 | |
| **Percent Time Saving** | 50.38% | |

The markers test repeatedly moves a marker along a trace and reads back the amplitude at the given frequency. This is similar to downloading a 10-point trace, but is meant to demonstrate the time saved in the common application of moving markers. This test had one of the highest time saving ratios.

## Test 7 - Integration (Total Power)

| | | |
|---|---|---|
| **Commands Used** | INIT:IMM | Take a single sweep |
| | CALC1:MARK1:MAX | Marker to peak |
| | CALC1:MARK1:SCEN | Marker to center |
| | CALC:MARK1:SRL | Marker to reference level |
| | CALC:MARK1:STAT OFF | Turn marker 1 off |
| | SENS:WAV:SPAN 10NM | Set the span |
| | SENS:BWID:RES 5NM | Set resolution bandwidth |
| | CALC1:TPOW:STAT 1 | Turn the total power state on |
| | CALC1:TPOW:DATA? | Query the total power |
| **Average Test Time (Display On) - ms** | 7213.3 | |
| **Average Test Time (Display Off) - ms** | 3751.3 | |
| **Percent Time Saving** | 47.99% | |

The integration test is meant to simulate finding the total power of an input source. The test first centers the display on the highest power peak, then runs a low-resolution sweep over the immediate area around that peak power. Because the display is changed several times during this process and the total power must be calculated, turning off the display yields a significant time savings.

## Test 8 - SMSR (Side Mode Suppression Ratio)

| | | |
|---|---|---|
| **Commands Used** | SENS:WAV:CENT 1550nm | Set center at 1550 nm |
| | SENS:WAV:SPAN 20nm | Set span to 20 nm |
| | INIT:IMM | Take a single sweep |
| | CALC:MARK:MAX | Place marker one at Max |
| | CALC:MARK:SCEN | Set marker to center |
| | CALC:MARK:SRL | Set the marker to reference level |
| | SENS:POW:DC:RANG:LOW –61DBM | Set the sensitivity to –61 dBm |
| | CALC:MARK1:Y? | Get the peak amplitude |
| | CALC:MARK1:MAX:NEXT | Set marker one to the next highest peak |
| **Average Test Time (Display On) - ms** | 9340.3 | |
| **Average Test Time (Display Off) - ms** | 5258.2 | |
| **Percent Time Saving** | 43.70% | |

The SMSR test is designed to use all of the commands in calculating the side mode suppression ratio on a laser source. The test centers on the largest signal, finds the peak power, and then determines the strength of the next highest peak. Calculating SMSR is a matter of dividing the first result by the second. This test is based on Example 10 in the 8614xB User's manual.

## Test 9 - OSNR (Optical Signal to Noise Ratio)

| | | |
|---|---|---|
| **Commands Used** | SENS:WAV:CENT 1550nm | Set center at 1550 nm |
| | SENS:WAV:SPAN 10nm | Set span to 10 nm |
| | INIT:IMM | Take a single sweep |
| | CALC:MARK:MAX | Place marker one at Max |
| | CALC:MARK:SCEN | Set marker to center |
| | CALC:MARK:SRL | Set the marker to reference level |
| | CALC:MARK2:MIN:LEFT | Find the local minimum to the left |
| | CALC:MARK3:MIN:RIGH | Find the local minimum to the right |
| | CALC:MARK1:Y? | Get the peak amplitude |
| | CALC:MARK2:Y? | Get the left pit amplitude |
| | CALC:MARK3:Y? | Get the right pit amplitude |
| | CALC:MARK1:X? | Get the peak wavelength |
| | CALC:MARK2:X? | Get the left pit wavelength |
| | CALC:MARK3:X? | Get the right pit wavelength |
| **Average Test Time (Display On) - ms** | 19910.5 | |
| **Average Test Time (Display Off) - ms** | 10722.5 | |
| **Percent Time Saving** | 46.15% | |

The OSNR test uses commands that are needed to calculate an optical signal to noise ratio using the interpolation technique. This involves finding the peak of the source, then finding the minimums on either side. The noise level is calculated by fitting a line to the two minimums and then finding the value of that line at the wavelength of the peak. The real time savings of having the display off for this test is over nine seconds. The high level functions *CALCulate:FUNCtion:OSNR:STATe ON* and *CALCulate:FUNCtion:OSNR:RESult?* can also be used and will correct the result for the noise bandwidth for a single channel. The DWDM application can be used to calculate OSNR for several channels.

## Test 10 - Trace Download

| | | |
|---|---|---|
| **Commands Used** | SENS:SWE:POIN 101 | Set trace length to 101 |
| | INIT:IMM | Take sweep |
| | FORM REAL | Set data format to real |
| | TRAC:DATA:Y? TRA | Request data |
| **Average Test Time (Display On) - ms** | 1070.5 | |
| **Average Test Time (Display Off) - ms** | 557.7 | |
| **Percent Time Saving** | 47.90% | |

The trace download test acquires a trace and then downloads it. Often times it is faster to download the entire trace and perform data calculations on the PC than it is to rely on the OSA to perform the measurements. Many PC processors are simply faster than the processor in the OSA. With the display turned off, the trace was acquired and retrieved in just over one-half second.

## Test 11 - Function

| | | |
|---|---|---|
| **Commands Used** | SENS:BWID:RES 10NM | Fix resolution BW |
| | INIT:IMM | Take a single sweep |
| | TRAC:FEED:CONT TRA, ALW | Continuously update trace A |
| | DISP:WIND:TRAC:STAT TRB,ON | Turn on trace B |
| | TRAC:FEED:CONT TRB, ALW | Continuously update trace B |
| | DISP:WIND:TRAC:STAT TRC,ON | Turn on trace C |
| | TRAC:FEED:CONT TRC, ALW | Continuously update trace C |
| | TRAC:FEED:CONT TRB,NEV | Freeze trace B |
| | INIT:CONT ON | Set up continuous sweep |
| | CALC3:MATH:EXPR ( TRA/TRB ) | Normalize trace A to B |
| | CALC3:MATH:STAT ON | Turn on normalization |
| **Average Test Time (Display On) - ms** | 6119.7 | |
| **Average Test Time (Display Off) - ms** | 2700.9 | |
| **Percent Time Saving** | 55.87% | |

The function test normalizes one trace relative to another. This test is based on Example 7 in the 8614xB User's manual. First three traces are turned on and acquired, then one is frozen as the reference, and finally the third is defined as the ratio of the first two. Since this process is graphic and calculation intensive, this test resulted in the greatest time saved as a percentage of any of the tests.

# Appendix B - Test Source Code

```
/*============================================================================
Program:        OSA benchmark
Author:         Agilent Technologies, LWD
Start Date:     13 April 2001
Last Modified:  17 April 2001
Description:    This program will be used to benchmark the 86145B in 'display off' remote operation versus
                'display on' operation

procedures tested:
   AutoAlign
   AutoMeasure
   Reset
   Zoom
   BW
   Trace download
   Trace Function
   SMSR
   OSNR
   Marker
   Integrate

Note: This program includes a simple user interface which was meant primarily for use during the development
      process.
============================================================================*/
//libraries
#include <windows.h>                    //windows library, required for GPIB/ENET interface
#include <stdio.h>                      //standarad io library
#include <stdlib.h>                     //standard C++ library
#include <decl-32.h>                    //GPIB header
#include <time.h>                       //time funcitons

//constants
#define FILENAME1 "c:\\my documents\\projects\\OSA bnechmark\\data.txt"
#define MAX_DATA                10      //the maximum number of tests allowed
//interface parameters
#define BDINDX                  0       // Board Index
#define PRIMARY_ADDR            23      // Primary address of device (default)
#define SECONDARY_ADDR          0       // Secondary address of device
#define TIMEOUT                 T30s    // Timeout value = 30 seconds
#define EOTMODE                 1       // Enable the END message
#define EOSMODE                 0       // Disable the EOS mode


//Prototypes
int which_test();                                                       //done
int get_repeat();                                                       //done
int get_display_setting();                                              //done
int setup_coms(int DisplaySetting);                                     //done
void run_tests(int instr, int repeat, int test, int *data);             //done
void pre_test(int instr);                                               //done
int test_AutoAlign(int instr);                                          //done
int test_AutoMeasure(int instr);                                        //done
int test_reset(int instr);                                              //done
int test_zoom(int instr);                                               //done
int test_BW(int instr);                                                 //done
int test_trace(int instr);                                              //done
int test_function(int instr);                                           //done
```

```
int test_SMSR(int instr);                                          //done
int test_OSNR(int instr);                                          //done
int test_markers(int instr);                                       //done
int test_integrate(int instr);                                     //done
void clean_up(int instr);                                          //done
void log_data(int data[], int repeat, int test, int dispOn);      //done
void write_IO(int instr, char cmd[], int size);                    //done
void read_IO(int instr, char rspns[], int size);                   //done
void Error_Code(char desc[]);                                      //done
void get_test_name(int test, char *name);                          //done
void run_again();                                                  //done
// void main()                                                     //done

/*============================================================================
Function:       Main
Description:    The main function controls program and data flow
Inputs:         none
Outputs:        none
============================================================================*/
void main(){
   //declarations
   //int test;              //a number specifying which test to run
   //int repeat;            //the number of times to repeat the test
   //int display;           //specifies whether the display is on or off
   int OSA;                 //specifies the OSA
   int data[MAX_DATA];      //an array for the time data and a pointer to it

   //test = which_test();                      //retrieve which test to run
   //repeat = get_repeat();                    //retrieve the number of repetitions
   //display = get_display_setting();          //retireve the display setting

   //automatically try each test the maximum number of times for (int Disp = 0; Disp <2; Disp++){
      OSA = setup_coms(Disp);                //set up the OSA
      for(int test = 1; test < 12; test++){
         printf("Test: %d, Disp: %d Run: ",test,Disp);
         run_tests(OSA, MAX_DATA, test, data); //run the tests
         log_data(data, MAX_DATA, test ,Disp); //log the data to a test file
      }//for(i)
   }//for (j)

   clean_up(OSA);                            //close communications
   printf("\nProgram complete!!\n");          //status report

   //run_again();          //ask the user if they want to re-run the program

}// main()
```

```
/*============================================================================
Function:       which_test
Description:    The which_test function queries the user for a number specifying which test to run
Inputs:         (none)
Outputs:        int - number specifying the desired test
                1  - Reset
                2  - AutoMeasure
                3  - AutoAlign
                4  - Zoom
                5  - BW meausrement
                6  - Markers
                7  - integrate
                8  - SMSR
                9  - OSNR
                10 - Trace download
                11 - Function
============================================================================*/
int which_test(){
  //declarations
   int tmp = 0;    //holds the user response

  //prompt the user
  printf ("Which test should be run?\n");
  printf (" 1.  Reset\n");
  printf (" 2.  AutoMeasure\n");
  printf (" 3.  AutoAlign\n");
  printf (" 4.  Zoom\n");
  printf (" 5.  BW Measurement\n");
  printf (" 6.  Markers\n");
  printf (" 7.  Integrate\n");
  printf (" 8.  SMSR\n");
  printf (" 9.  OSNR\n");
  printf (" 10. Trace Download\n");
  printf (" 11. Function\n");
  scanf("%d", &tmp);

  if ((tmp < 12) && (tmp > 0))          //check for valid input
     return tmp;                        //return valid input
  else{
      printf("sorry, '%d' is not a choice\n", tmp);      //notify user of error
      return which_test();                               //recursively re-prompt user
  }//else

}//which_test()

/*============================================================================
Function:       get_repeat
Description:    The which_test function queries the user for the number of times to repeat the test
Inputs:         (none)
Outputs:        int - number specifying the number of repetitions
```

```
==========================================================================*/
int get_repeat(){
  //declarations
   int tmp = 0;   //holds the user response

  //prompt the user
  printf ("\nHow many times should it be repeated? (1 to %d)\n", MAX_DATA);
  scanf("%d", &tmp);

  if ((tmp < MAX_DATA) && (tmp > 0))    //check for valid input
     return tmp;                        //return valid input
  else{
      printf("sorry, '%d' is out of range\n", tmp);      //notify user of error
      return get_repeat();                               //recursively re-prompt user
  }//else
}//get_repeat()

/*==========================================================================
Function:       get_display_setting
Description:    The which_test function queries the user for whither the display will be on or off
Inputs:         (none)
Outputs:        int - 1 -> display on
                      0 -> dipslay off
==========================================================================*/
int get_display_setting(){
    //declarations
   int tmp = 0;   //holds the user response

  //prompt the user
  printf ("\nShould the display be turned ON?\n");
  printf (" 0.  OFF\n");
  printf (" 1.  ON\n");
  scanf("%d", &tmp);

  if ((tmp < 2) && (tmp > -1))      //check for valid input
     return tmp;                    //return valid input
  else{
      printf("sorry, '%d' is not a choice\n", tmp);      //notify user of error
      return get_display_setting();                      //recursively re-prompt user
  }//else
}//get_display_setting()


/*==========================================================================
Function:       run_again
Description:    The run_again method asks the user if they want to re-run the program and then recursisvely calls
                the program
Inputs:         (none)
Outputs:        (none)
```

```
==========================================================================*/
void run_again(){
    //declarations
   int tmp = 0;   //holds the user response

  //prompt the user
  printf ("\nRe-run the program?\n");
  printf (" 1.  YES\n");
  scanf("%d", &tmp);

  if (tmp == 1)   //re-run the program for 1
     main();

}//run_again()


/*==========================================================================
Function:       setup_coms
Description:    sets up the communications with the OSA and puts the chooses the appropriate settings
Inputs:         (none)
Outputs:        int - number designating the insturment. (zero returned for coms failure)
==========================================================================*/
int setup_coms(int DisplaySetting){
  int OSA = 0;            //temp variable which designates the OSA
  int crntDisp;           //stores the current display setting
  char buffer[256];       //stores returned data from the OSA
  char cmd[8];            //stores the display on or off command

  //printf("\nSetting up OSA\n");  //report status to user

  OSA = ibdev(BDINDX, PRIMARY_ADDR, SECONDARY_ADDR, TIMEOUT, EOTMODE, EOSMODE);

  write_IO(OSA, "*RST\n", 5);       //Reset the instrument
  write_IO(OSA, "*OPC?\n", 5);      //query for completion
  read_IO(OSA, buffer, 255);        // read response

  write_IO(OSA, "*CLS\n", 5);       //clear the status registers

  write_IO(OSA, "DISP?\n", 5);      //query for display setting
  read_IO(OSA, buffer, 255);        // read response

  sscanf(buffer, "%d", &crntDisp);          //parse the returned string

  if (crntDisp != DisplaySetting){          //change the display only if needed
     sprintf(cmd, "DISP %d\n", DisplaySetting);             //create the command

     write_IO(OSA, cmd, 7);                                 //toggles the display setting
     write_IO(OSA, "*OPC?\n", 5);                           //query for completion
     read_IO(OSA, buffer, 255);                             // read response
  }//if(crntDisp != DisplaySetting)

  write_IO(OSA, "SYST:COMM:GPIB:BUFF ON\n", 23);  //turn on GPIB buffer

  //printf("Set up complete\n");                  //report status to user

  return OSA;

}//setup_coms()
```

```
/*============================================================================
Function:       run_tests
Description:    runs the specified test the specified number of times
Inputs:         int instr - designates the OSA
                int repeat - the number of times the test will be repeated
                int test - specifies whihc test to run
                int *data - a pointer to the data output array
Outputs:        (none)
============================================================================*/
void run_tests(int instr, int repeat, int test, int *data){
   for(int i = 0; i < repeat; i++){
      pre_test(instr);                  //reset the insturment before each test

      printf("%d", i+1);
      switch(test){                     //run the specified test
      case 1:                           //reset
         data[i] = test_reset(instr);
         break;
      case 2:                           //automeasure
         data[i] = test_AutoMeasure(instr);
         break;
      case 3:                           //autoalign
         data[i] = test_AutoAlign(instr);
         break;
      case 4:                           //zoom
         data[i] = test_zoom(instr);
         break;
      case 5:                           //Bandwidth
         data[i] = test_BW(instr);
         break;
      case 6:                           //markers
         data[i] = test_markers(instr);
         break;
      case 7:                           //integration
         data[i] = test_integrate(instr);
         break;
      case 8:                           //SMSR
         data[i] = test_SMSR(instr);
         break;
      case 9:                           //OSNR
         data[i] = test_OSNR(instr);
         break;
      case 10:                          //Trace Download
         data[i] = test_trace(instr);
         break;
      case 11:                          //function
         data[i] = test_function(instr);
         break;
      }//switch(test)

   }//for()

}//run_tests()
```

```
/*============================================================================
Function:       pre_test
Description:    runs the required commands before each test (*rst, etc.)
Inputs:         int instr - designates the OSA
Outputs:        (none)
============================================================================*/
void pre_test(int instr){

   char buffer[255];

   write_IO(instr, "*RST\n", 5);       //Reset the instrument
   write_IO(instr, "*OPC?\n", 5);      //query for completion
   read_IO(instr, buffer, 255);        //read response

}//pre_test()

/*============================================================================
Function:       test_reset
Description:    the test_reset function measure the time required for a instrument reset
Inputs:         int instr - an integer which represents the instrument
Outputs:        int - the test time in ms

Commands used:
Command                 Use
*RST                    instrument reset
============================================================================*/
int test_reset(int instr){
   char buffer[255];
   int start = clock();                //record start time

   write_IO(instr, "*RST;*OPC?\n", 10);  //Reset the instrument
   read_IO(instr, buffer, 255);          //read response

   return (clock()-start);               //return the elapsed time
}//test_reset()

/*============================================================================
Function:       test_AutoAlign
Description:    the test_AutoAlign function measure the time required for an AutoAlign
Inputs:         int instr - an integer which represents the instrument
Outputs:        int - the test time in ms

Commands used:
Command                 Use
CAL:ALIG:AUTO           AutoAlign
============================================================================*/
int test_AutoAlign(int instr){
    char buffer[255];
    int start = clock();                          //record start time

    write_IO(instr, "CAL:ALIG:AUTO;*OPC?\n", 19);  //AutoAlign
    read_IO(instr, buffer, 255);                   //read response

    return (clock()-start);                        //return the elapsed time
}//test_AutoAlign()
```

```
/*============================================================================
Function:       test_AutoMeasure
Description:    the test_AutoMeasure function measure the time required for an AutoMeasure
Inputs:         int instr - an integer which represents the instrument
Outputs:        int - the test time in ms

Commands used:
Command                                 Use
DISP:WIND:TRAC:ALL:SCAL:AUTO    AutoMeasure
============================================================================*/
int test_AutoMeasure(int instr){
   char buffer[255];
   int start = clock();                         //record start time

   write_IO(instr, "DISP:WIND:TRAC:ALL:SCAL:AUTO;*OPC?\n", 35);
                                                //AutoMeasure
   read_IO(instr, buffer, 255);                 //read response

   return (clock()-start);                      //return the elapsed time
}//test_AutoMeasure()

/*============================================================================
Function:       test_zoom
Description:    the test_zoom function measure the time required to change the wavelength limits so that the peak
                value is centered
Inputs:         int instr - an integer which represents the instrument
Outputs:        int - the test time in ms

Commands:
Command                 Use
INIT:IMM                run a single sweep
CALC1:MARK1:MAX         Marker to peak power
CALC1:MARK1:SCEN        marker to center
SENS:WAV:SPAN 10nm      set the wavelength span to 10nm
CALC1:MARK1:X?          get marker wavelength
CALC2:MARK1:Y?          get marker amplitude
============================================================================*/
int test_zoom(int instr){
   char buffer[256];        //temp buffer
   int start = clock();     //start time

   write_IO(instr, "INIT:IMM;*OPC?\n", 15);             //Trigger a sweep
   read_IO(instr, buffer, 255);                         //read response

   write_IO(instr, "CALC1:MARK1:MAX\n", 16);            //set marker to peak value
   write_IO(instr, "CALC1:MARK1:SCEN\n", 17);           //center on the marker
   write_IO(instr, "SENS:WAV:SPAN 10nm\n", 19);         //set the WL span to 10nm

   write_IO(instr, "INIT:IMM;*OPC?\n", 15);             //re - sweep
   read_IO(instr, buffer, 255);                         //read response

   write_IO(instr, "CALC1:MARK1:X?\n", 15);             //get marker wavelength
   read_IO(instr, buffer, 255);                         //read response

   write_IO(instr, "CALC1:MARK1:Y?\n", 15);             //get marker amplitude
   read_IO(instr, buffer, 255);                         //read response

   return (clock()-start);
}//test_zoom()
```

```
/*============================================================================
Function:       test_BW
Description:    the test_BW function measure the time required to measure BW
Inputs:         int instr - an integer which represents the instrument
Outputs:        int - the test time in ms

Commands
Command                                 Use
sens:wav:star 1530nm                    Set start wavelength
sens:wav:stop 1570nm                    Set stop Wavelength
sens:pow:dc:rang:low -60dBm             Set sensitivity
init:imm                                Take Sweep
calc1:mark1:max                         Marker to peak
calc1:mark1:scen                        Marker to center
calc1:mark1:x?                          Read marker wavelength
calc1:mark1:y?                          Read marker amplitude
sens:bwid:res 0.1 nm                    set resolution bandwidth to min
sens:wav:span 2nm                       Set span to highest resolution
calc1:mark1:func:bwid:ndb -20.0 db      Select db down where bw is calculated
calc1:mark1:func:bwid:int on            Enable bw marker interpolation
calc1:mark1:func:bwid:read wav          Sets the BW unit of measurement to WL
calc1:mark1:func:bwid:stat on           Enable bandwidth marker
calc1:mark1:func:bwid:res?              Returns axis values between markers
============================================================================*/
int test_BW(int instr){
   char buffer[256];          //temp buffer
   int start = clock();       //start time

   int tmp = 0;

   write_IO(instr, "SENS:WAV:STAR 1530nm\n",21);              //set start WL
   write_IO(instr, "SENS:WAV:STOP 1570nm\n",21);              //set stop WL
   write_IO(instr, "SENS:POW:DC:RANG:LOW -60dBm\n",28);       //set sensitivity
   write_IO(instr, "INIT:IMM;*OPC?\n", 15);                   //Trigger a sweep
   read_IO(instr, buffer, 255);                               //read response

   write_IO(instr, "CALC1:MARK1:MAX\n", 16);                  //set marker to peak value
   write_IO(instr, "CALC1:MARK1:SCEN\n", 17);                 //center on the marker

   write_IO(instr, "SENS:BWID:RES 0.1nm\n", 20);              //set res bandwidth
   write_IO(instr, "SENS:WAV:SPAN 2nm\n", 18);                //set span

   write_IO(instr, "INIT:IMM;*OPC?\n", 15);                   //re - sweep
   read_IO(instr, buffer, 255);                               //read response

   write_IO(instr, "calc1:mark1:max\n",16);                   //marker to max
   write_IO(instr, "calc1:mark1:func:bwid:ndb -20.0\n",30);   //set bw power
   write_IO(instr, "calc1:mark1:func:bwid:int on\n",29);      //enable BW marker interpolation
   write_IO(instr, "calc1:mark1:func:bwid:read wav\n",31);    //measure BW by WL
   write_IO(instr, "calc1:mark1:func:bwid:stat on\n",30);     //enable BW markers
   write_IO(instr, "calc1:mark1:func:bwid:res?\n", 27);       //get the BW
   read_IO(instr, buffer, 255);                               //read response

   return (clock()-start);
}//test_BW()
```

```
/*============================================================================
Function:       test_Markers
Description:    the test_Markers function measure the time required to place and recover ten markers
Inputs:         int instr - an integer which represents the instrument
Outputs:        int - the test time in ms

Commands
 Command                        Use
SENS:WAV:STAR 1480nm            Set start wavelength
SENS:WAV:STOP 1580nm            Set stop Wavelength
INIT:IMM                        run a single sweep
CALC:MARK:X:WAV 1480nm          sets the marker position
CALC:MARK:Y?                    reads the amplitude at the marker position
============================================================================*/
int test_markers(int instr){
  char buffer[256];
  char cmd[50];                 //stores the marker placing command
  int start = clock();

   write_IO(instr,"SENS:WAV:STAR 1480nm\n",21);     //Set start wavelength
   write_IO(instr,"SENS:WAV:STOP 1580nm\n",21);     //Set stop Wavelength
   write_IO(instr,"INIT:IMM;*OPC?\n",15);           //run a single sweep
   read_IO(instr, buffer, 255);                     //read response

   for(int i = 1480; i < 1590; i = i+10){
      sprintf(cmd,"CALC:MARK:X:WAV %dnm\n",i);      //build the command
      write_IO(instr,cmd,23);                       //sets the marker position
      write_IO(instr,"CALC:MARK:Y?\n",13);          //reads the marker amplitude
      read_IO(instr, buffer, 255);                  //read response
  }//for()

   return (clock()-start);
}//test_markers()

/*============================================================================
Function:       test_integrate
Description:    the test_integrate function measure the time required to integrate a trace
Inputs:         int instr - an integer which represents the instrument
Outputs:        int - the test time in ms

Commands:
 Command                Use
init:imm                Take a single sweep
calc1:mark1:max         Marker to peak
calc1:mark1:scen        Marker to center
calc:mark1:srl          marker to reference level
calc:mark1:stat off     Turn marker 1 off
sens:wav:span 10nm      Set the Span
sens:bwid:res 5nm       Set resolution bandwidth
calc1:tpow:stat 1       turn the tpower state on
calc1:tpow:data?        Query the total power
```

```
==========================================================================*/
int test_integrate(int instr){
   char buffer[256];
   int start = clock();

   write_IO(instr,"INIT:IMM;*OPC?\n",15);          //run a single sweep
   read_IO(instr, buffer, 255);                    //read response

   write_IO(instr,"calc1:mark1:max\n",16);         //Marker to peak
   write_IO(instr,"calc1:mark1:scen\n",17);        //Marker to center
   write_IO(instr,"calc:mark1:srl\n",15);          //marker to reference level
   write_IO(instr,"calc:mark1:stat off\n",20);     //Turn marker 1 off
   write_IO(instr,"SENS:wav:span 10nm\n",19);      //Set the Span
   write_IO(instr,"sens:bwid:res 5nm\n",18);       //Set resolution bandwidth


   write_IO(instr,"INIT:IMM;*OPC?\n",15);          //run a single sweep
   read_IO(instr, buffer, 255);                    //read response

   write_IO(instr,"calc1:tpow:stat 1\n",18);       //turn the tpower state on
   write_IO(instr,"calc1:tpow:data?\n",17);        //Query the total power
   read_IO(instr, buffer, 255);                    //read response

   return (clock() - start);
}//test_integrate()

/*==========================================================================
Function:      test_SMSR
Description:   the test_SMSR function measure the time required for a SMSR measurement
Inputs:        int instr - an integer which represents the instrument
Outputs:       int - the test time in ms

Commands:
   Command                          Use
SENS:WAV:CENT 1550nm                Set center at 1550
SENS:WAV:SPAN 20nm                  Set span to 20nm
INIT:IMM                            Take a single sweep
CALC:MARK:MAX                       Place marker one at Max
CALC:MARK:SCEN                      Set marker to center
CALC:MARK:SRL                       Set the marker to reference level
SENS:POW:DC:RANGe:LOW -61DBM        Set the sensitivity to -61dBm
CALC:MARK1:Y?                       Get the peak amplitude
CALC:MARK1:MAX:NEXT                 Set mark one to the next highest peak
```

```
==========================================================================*/
int test_SMSR(int instr){
   char buffer[256];
   int start = clock();

   write_IO(instr, "SENS:WAV:CENT 1550nm\n", 21);              //set center WL
   write_IO(instr, "SENS:WAV:SPAN 20nm\n", 19);                //set span

   write_IO(instr, "INIT:IMM;*OPC?\n", 15);                    //Sweep Once
   read_IO(instr, buffer, 255);                                //read response

   write_IO(instr, "CALC:MARK:MAX\n", 14);                     //set marker1 to the peak
   write_IO(instr, "CALC:MARK:SCEN\n", 15);                    //center on the marker
   write_IO(instr, "CALC:MARK:SRL\n", 14);                     //set the reference level
   write_IO(instr, "SENS:POW:DC:RANGe:LOW -61DBM\n", 29);      //set the sensitivity

   write_IO(instr, "INIT:IMM;*OPC?\n", 15);                    //Sweep Once
   read_IO(instr, buffer, 255);                                //read response

   write_IO(instr, "CALC:MARK1:MAX\n", 15);                    //set marker1 to the peak
   write_IO(instr, "CALC:MARK1:Y?\n", 14);                     //get the peak amplitude
   read_IO(instr, buffer, 255);                                //read response

   write_IO(instr, "CALC:MARK1:MAX:NEXT\n", 20);               //find the next nighest peak
   write_IO(instr, "CALC:MARK1:Y?\n", 14);                     //get the peak amplitude
   read_IO(instr, buffer, 255);                                //read response

   return (clock()-start);
}//test_SMSR()

/*==========================================================================
Function:       test_OSNR
Description:    the test_OSNR function measure the time required for a ONSR measurement
Inputs:         int instr - an integer which represents the instrument
Outputs:        int - the test time in ms

Commands
   Command                  Use
SENS:WAV:CENT 1550nm        Set center at 1550
SENS:WAV:SPAN 10nm          Set span to 10nm
INIT:IMM                    Take a single sweep
CALC:MARK:MAX               Place marker one at Max
CALC:MARK:SCEN              Set marker to center
CALC:MARK:SRL               Set the marker to reference level
CALC:MARK2:MIN:LEFT         Find the local minimum to the left
CALC:MARK3:MIN:RIGH         Find the local minimum to the right
CALC:MARK1:Y?               Get the peak amplitude
CALC:MARK2:Y?               Get the left pit amplitude
CALC:MARK3:Y?               Get the right pit amplitude
CALC:MARK1:X?               Get the peak Wavelength
CALC:MARK2:X?               Get the left pit Wavelength
CALC:MARK3:X?               Get the right pit Wavelength
```

```
==========================================================================*/
int test_OSNR(int instr){
   char buffer[256];
   int start = clock();

   write_IO(instr,"SENS:WAV:CENT 1550nm\n",21);      //Set center at 1550
   write_IO(instr,"SENS:WAV:SPAN 20nm\n",19);       //Set span to 10nm

   write_IO(instr,"INIT:IMM;*OPC?\n",15);           //Take a single sweep
   read_IO(instr, buffer, 255);                     //read response

   write_IO(instr,"CALC:MARK:MAX\n",14);            //Place marker one at Max
   write_IO(instr,"CALC:MARK:SCEN\n",15);           //Set marker to center
   write_IO(instr,"CALC:MARK:SRL\n",14);            //Set the marker to ref level

   write_IO(instr,"INIT:IMM;*OPC?\n",15);           //Take a single sweep
   read_IO(instr, buffer, 255);                     //read response

   write_IO(instr,"CALC:MARK2:MIN:LEFT\n",20);      //Find the left local minimum
   write_IO(instr,"CALC:MARK3:MIN:RIGH\n",20);      //Find the right local minimum

   write_IO(instr,"CALC:MARK1:Y?\n",14);            //Get the peak amplitude
   read_IO(instr, buffer, 255);                     //read response

   write_IO(instr,"CALC:MARK2:Y?\n",14);            //Get the left pit amplitude
   read_IO(instr, buffer, 255);                     //read response

   write_IO(instr,"CALC:MARK3:Y?\n",14);            //Get the right pit amplitude
   read_IO(instr, buffer, 255);                     //read response

   write_IO(instr,"CALC:MARK1:X?\n",14);            //Get the peak Wavelength
   read_IO(instr, buffer, 255);                     //read response

   write_IO(instr,"CALC:MARK2:X?\n",14);            //Get the left pit Wavelength
   read_IO(instr, buffer, 255);                     //read response

   write_IO(instr,"CALC:MARK3:X?\n",14);            //Get the right pit Wavelength
   read_IO(instr, buffer, 255);                     //read response

   return(clock()-start);
}//test_OSNR

/*==========================================================================
Function:       test_trace
Description:    the test_trace function measure the time required to downlaod a trace
Inputs:         int instr - an integer which represents the instrument
Outputs:        int - the test time in ms

Commands
Command                 Use
sens:swe:poin 101       Set trace length to 101
init:imm                Take sweep
form real               Set data format to real
trac:data:y? tra        Request data
```

```
==========================================================================*/
int test_trace(int instr){
    char buffer[820]; //buffer must be big enough for all of the trace data

  int tmp = test_zoom(instr);                  //use'test_zoom' to set up the OSA

  int start = clock();                         //record start time

  write_IO(instr, "SENS:SWE:POIN 101\n", 18);  //set to 101 trace pts
  write_IO(instr, "FORM REAL\n", 10);          //Real data format

  write_IO(instr, "INIT:IMM;*OPC?\n", 15);     //sweep
  read_IO(instr, buffer, 255);                 //read response

  write_IO(instr, "TRAC:DATA:Y? TRA\n", 17);   //query trace data
  read_IO(instr, buffer, 820);                 //read response

  return (clock()-start);                      //return the elapsed time
}//test_trace

/*==========================================================================
Function:     test_Function
Description:  the test_Function function measure the time required to normalize a trace
Inputs:       int instr - an integer which represents the instrument
Outputs:      int - the test time in ms

Commands:
   Command                  Use
Sens:bwid:res 10nm          Fix resolution bw
init:imm                    Take a single sweep
Trac:Feed:Cont TrA, Alw     continuously update trace A
disp:Wind:Trac:Stat TrB,ON  Turn on Trace B
Trac:Feed:Cont TrB, Alw     continuously Update trace B
disp:Wind:Trac:Stat TrC,ON  Turn on Trace C
Trac:Feed:Cont TrC, Alw     continuously Update trace B
Trac:Feed:Cont TrB,Nev      Freeze trace B
init:cont on                Set up continuous sweep
Calc3:Math:Expr ( TRA/TRB ) Normalize Trace A to B
Calc3:Math:Stat On          Turn on normalization
```

```
==========================================================================*/
int test_function(int instr){
   char buffer[256];
   int start = clock();

   write_IO(instr, "SENS:BWID:RES 10nm\n", 19);          // set the res bandwidth

   write_IO(instr, "INIT:IMM;*OPC?\n", 15);              //Trigger a sweep
   read_IO(instr, buffer, 255);                          //read response

   write_IO(instr, "TRAC:FEED:CONT TRA, ALW\n", 24);     //always update trace A
   write_IO(instr, "DISP:WIND:TRAC:STAT TRB,ON\n", 27);  //turn trace B on
   write_IO(instr, "TRAC:FEED:CONT TRB, ALW\n", 24);     //always update trace B
   write_IO(instr, "DISP:WIND:TRAC:STAT TRC,ON\n", 27);  //turn trace C on
   write_IO(instr, "TRAC:FEED:CONT TRC, ALW\n", 24);     //always update trace C
   write_IO(instr, "TRAC:FEED:CONT TRB, NEV\n", 24);     //Freeze trace B
   write_IO(instr, "CALC3:MATH:EXPR (TRA/TRB)\n", 26);   //C = A - B
   write_IO(instr, "CALC3:MATH:STAT ON\n", 19);          //turn on math

   write_IO(instr, "INIT:CONT ON\n", 13);                //turn on continuous sweep

   return (clock()-start);
}//test_function

/*============================================================================
Function:       clean_up
Description:    the clean_up function makes sure that the display is turned back on and that the instrument is
                returned to local control.
Inputs:         int instr - an integer which represents the instrument
Outputs:        int - the test time in ms
============================================================================*/
void clean_up(int instr){

   char buffer[256];
   printf("cleaning up\n");              //status report

   write_IO(instr, "DISP ON\n", 8);      //turn the display on
   write_IO(instr, "*OPC?\n", 5);        //query for completion
   read_IO(instr, buffer, 255);          //read response

   ibloc(instr);                         //set OSA to local
   ibonl(instr, 0);                      //take OSA off line

   printf("clean up complete\n");        //status report
}//clean_up()

/*============================================================================
Function:       log_data
Description:    The log_data function outputs the data to a CSV text file.
                it also outputs the name of the test and whether the display was on or not.  Finally, it calculates the
                average time and outputs that.  All of the data is output onto a single line and appended to the
                existing file.  this allows multiple tests to write to the same file.
Inputs:         int data[] - the time values of the tests run
                int repeat - the number tests which were run
                int test - the number of the test which was run
                int dispOn - whether the display was on
Outputs:        none
```

```
==========================================================================*/
void log_data(int data[], int repeat, int test, int dispOn){
   FILE *fp;                                                  //file pointer
   char test_name[16];                                        //the name of the test which was run
   char disp_str[4];                                          //whether the display was on
   char csv_data[256];                                        //output buffer
   double avg = 0 ;                                           //the calculated average of the data points
   sprintf(csv_data,"\0");                                    //initialze the data string

   printf("\nwritting file\n");                               //status report

   get_test_name(test,test_name);                             //convert the test number to a name string

   if(dispOn == 1)                                            //convert the dispOn arg to a null terminated string
      sprintf(disp_str, "ON\0");
   else
      sprintf(disp_str, "OFF\0");

   for(int i=0;i<repeat;i++){                                 //step through the data and.....
      avg = avg + data[i];                                    //sum the data
      sprintf(csv_data,"%s,%d\0",csv_data,data[i]);           //add point to the string
   }
   avg = avg / repeat;                                        //calculate the average

   fp = fopen(FILENAME1, "a");                                //open file in append mode
   fprintf(fp,"%s%s%s,%f\n",test_name,disp_str,csv_data,avg); //construct the output string and send it to the file
   fclose(fp);                                                //close file

}//log_data()

/*==========================================================================
Function:      get_test_name
Description:   this function simply converts the integer representation to a
               short description of a test
Inputs:        int test - the number of the test
               char *name - pointer to the output string
Outputs:       none
```

```
==========================================================================*/
void get_test_name(int test, char *name){

   switch(test){                        //run the specified test
      case 1:                           //reset
         sprintf(name,"Reset,\0");
         break;
      case 2:                           //automeasure
         sprintf(name,"AutoMeasure,\0");
         break;
      case 3:                           //autoalign
         sprintf(name,"AutoAlign,\0");
         break;
      case 4:                           //zoom
         sprintf(name,"Zoom,\0");
         break;
      case 5:                           //Bandwidth
         sprintf(name,"Bandwidth,\0");
         break;
      case 6:                           //markers
         sprintf(name,"Markers,\0");
         break;
      case 7:                           //integration
         sprintf(name,"Integration,\0");
         break;
      case 8:                           //SMSR
         sprintf(name,"SMSR,\0");
         break;
      case 9:                           //OSNR
         sprintf(name,"OSNR,\0");
         break;
      case 10:                          //trace download
         sprintf(name,"Trace Download,\0");
         break;
      case 11:                          //function
         sprintf(name,"Function,\0");
         break;
   }//switch(test)
}
/*==========================================================================
Function:       write_IO
Description:    The Write_IO sub combines the board level write function and the error check.  The reason for this
                short sub is due to simplify the code as these functions are almost always used together.
Inputs:         int instr - an integer which represents the instrument
                char cmd[] - the command to be written
                int size - the length of the command string
Outputs:        none
==========================================================================*/
void write_IO(int instr, char cmd[], int size){
  ibwrt (instr, cmd, size);    //querry for the id number
  if (ibsta & ERR) Error_Code(strcat("could not write command: ",cmd));
                                       //check for errors
}
```

```
/*============================================================================
Function:       read_IO
Description:    The read_IO sub combines the board level read function and the error check.  The reason for this short
                sub is due to simplify the code as these functions are almost always used together.
Inputs:         int instr - an integer which represents the instrument
                char rspns[] - the target loacation of the returned data
                int size - the number of caharacters ot be read back
Outputs:        none
============================================================================*/
void read_IO(int instr, char rspns[], int size){
  ibrd (instr, rspns, size);                    //querry for the id number
  if (ibsta & ERR) Error_Code("did not recieve response");
                                                //check for errors
}

/*============================================================================
Function:       Error_Code
Description:    The Error_Code function displays an error message and stops the pogram if any errors are
                encountered
Inputs:         char[] desc - a description of the error
Outputs:        none
============================================================================*/

void Error_Code(char desc[]){

 //GPIB Error Codes
 char ErrorCodes[21][5] = {"EDVR", "ECIC", "ENOL", "EADR", "EARG",
                           "ESAC", "EABO", "ENEB", "EDMA", "",
                           "EOIP", "ECAP", "EFSO", "", "EBUS",
                           "ESTB", "ESRQ", "", "", "", "ETAB"};

 printf("Error : %s\nibsta = 0x%x iberr = %d (%s)\n",
         desc, ibsta, iberr, ErrorCodes[iberr]);

 exit(1);
} //Error_Code()
```

# Appendix C - Test Setup

### PC set up

- 450 MHz Intel Pentium® III processor with 512 KB Cache and 128 MB of Ram
- Windows® NT 4.0 (Service Pack 4)
- National Instruments PCI-GPIB card with NI 488.2 Version 1.6 (August 1999) drivers

### OSA

Agilent 86145B with vB.04.00 (prototype) firmware

### Source

Agilent 83403A 1550nm FP laser source.
The laser source remained on and unmodulated for all of the tests and was fed directly into the OSA input via a 40 cm, 9/125 connector fiber

---

Pentium is a U.S. registered trademark of Intel Corporation.
Windows NT is a U.S. registered trademark of Microsoft Corporation.

## Agilent Technologies'
## Test and Measurement Support, Services, and Assistance

Agilent Technologies aims to maximize the value you receive, while minimizing your risk and problems. We strive to ensure that you get the test and measurement capabilities you paid for and obtain the support you need. Our extensive support resources and services can help you choose the right Agilent products for your applications and apply them successfully. Every instrument and system we sell has a global warranty. Support is available for at least five years beyond the production life of the product. Two concepts underlie Agilent's overall support policy: "Our Promise" and "Your Advantage."

### Our Promise

Our Promise means your Agilent test and measurement equipment will meet its advertised performance and functionality. When you are choosing new equipment, we will help you with product information, including realistic performance specifications and practical recommendations from experienced test engineers. When you use Agilent equipment, we can verify that it works properly, help with product operation, and provide basic measurement assistance for the use of specified capabilities, at no extra cost upon request. Many self-help tools are available.

### Your Advantage

Your Advantage means that Agilent offers a wide range of additional expert test and measurement services, which you can purchase according to your unique technical and business needs. Solve problems efficiently and gain a competitive edge by contracting with us for calibration, extra-cost upgrades, out-of-warranty repairs, and on-site education and training, as well as design, system integration, project management, and other professional engineering services. Experienced Agilent engineers and technicians worldwide can help you maximize your productivity, optimize the return on investment of your Agilent instruments and systems, and obtain dependable measurement accuracy for the life of those products.

**By internet, phone, or fax, get assistance with all your test & measurement needs.**

**Online assistance:**
**www.agilent.com/comms/lightwave**


**Phone or Fax**
**United States:**
(tel) 1 800 452 4844

**Canada:**
(tel) 1 877 894 4414
(fax) (905) 282 6495

**Europe:**
(tel) (31 20) 547 2323
(fax) (31 20) 547 2390

**Japan:**
(tel) (81) 426 56 7832
(fax) (81) 426 56 7840

**Latin America:**
(tel) (305) 269 7500
(fax) (305) 269 7599

**Australia:**
(tel) 1 800 629 485
(fax) (61 3) 9210 5947

**New Zealand:**
(tel) 0 800 738 378
(fax) 64 4 495 8950

**Asia Pacific:**
(tel) (852) 3197 7777
(fax) (852) 2506 9284




Printed in USA June 1, 2001
5988-2918EN


Agilent Technologies