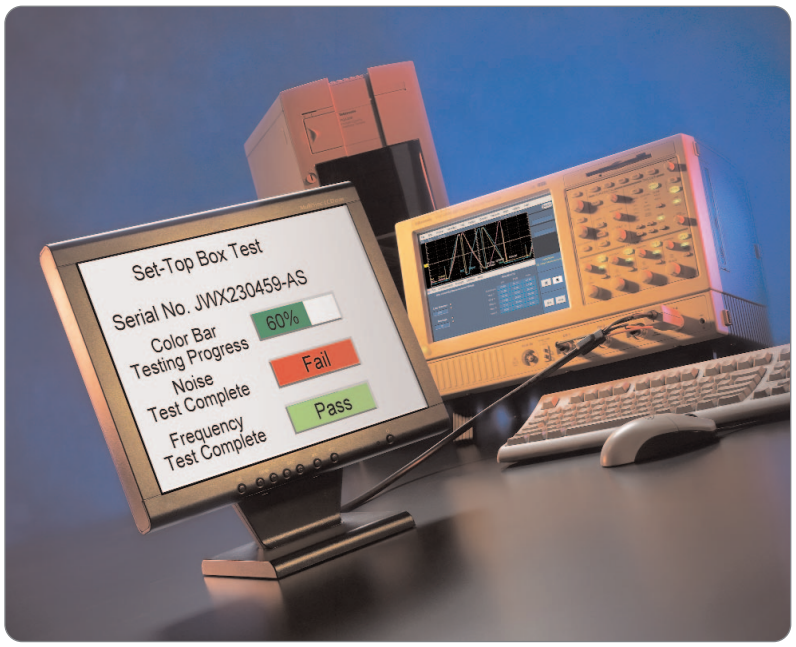
Using the VM5000HD and the GPIB Interface



▶ Automated testing of many consumer products within a manufacturing environment requires software control of the measurement instrument via GPIB.

Consumer video equipment is now commonplace in most homes and is almost part of the furniture. The evolution of video equipment, which offers, improved picture performance that is achieved by extended definition television and is available on many progressive Digital Versatile Disk players (480p & 576p). Further increased resolution is offered on High Definition (HD) capable equipment and is accessible via terrestrial or satellite services. These new types of video equipment require manufactures to develop new testing methodologies to ensure the quality of their product.

Manufacturers of consumer video equipment need to ensure that the video outputs of their consumer product meet video specifications and that the device is functioning correctly. To ensure the performance of the consumer device various video measurements are performed on the video output of the device under test (DUT). These measurements need to be made quickly and accurately on the DUT in order to allow a fast throughput of product on the production line. Test routines are normally operated automatically and provide operators with simple pass or fail warnings at the completion of the tests. To perform this test the software program controls various measurement instruments, this is normally done via a GPIB (General Purpose Interface Bus IEEE 488) connection.

The VM5000HD is an analog High Definition (HD) automated measurement instrument that performs 6 different types of video measurements on various HD standards (1080i 59.94, 1080i 50 & 720p 59.94) and both progressive extended definition video formats (480p & 576p). The measurements performed by the instrument are detailed in a Tektronix application note (Literature Number 25W-16653-0) Analog High Definition Component Measurement. This instrument can be controlled via a GPIB command set and allows the functionality of the instrument to be manipulated by a software program. An example software program is provided for the VM5000HD that illustrates the various functions needed to perform an automated routine. This program can be tailored to meet your individual requirements and this application note details how each part of these functions work on the VM5000HD.

Tektronix
Enabling Innovation

There are six different automated measurements performed by the VM5000HD that allow a DUT to be characterized and ensure conformance of the device in either RGB or YPbPr formats. The following measurements are available in the VM5000HD:

Channel Delay	Compares the inter channel timing relationship of each channel and provides a value for the time in nanoseconds that one channel is advanced or delayed from the other. The ideal value would be 0.0ns for each comparison. A positive value indicates a channel delay of 'A' with respect to 'B' and a negative value indicates a channel advance of 'A' with respect to 'B'.
Color Bars	Provides amplitude measurements for each of the three channels. A total of eight amplitude measurements are made on each channel corresponding to each color of the color bar signal. Limits are typically used to quantify the range of acceptable performance of the device.
Non-Linearity	Amplification stages should amplify each amplitude value equally. If the system does not process the signal correctly this can lead to different gains being applied to the signal at different amplitude levels and produce a non-linearity in the performance of the device.
Noise Measurement	Noise is inherent in every electrical circuit but excessive noise can become visible in the picture at approximately 40dB. It can provide a simple indication of some failure within the device.
Multiburst	This measurement allows the user to check the frequency response of the system. A low frequency flag is compared in amplitude to various increasing frequency packets of ideally the same amplitude and the deviation in amplitude between the flag and the frequency packet is measured in dB.
Sync Timing	The various video standards have different timings for each of the parameters within the horizontal blanking interval. There are specific tolerances that each format is required to meet.

Within the manufacturing environment it is important for these measurements to be made on the DUT to be within acceptable limits for conformance to the specifications of the device. It is therefore necessary to define a set of parameters for each measurement to provide a pass or fail indication. Within the main.c program the following tolerances.h file is used.

```
/*
 * tolerances.h version 1.0
 *
 * This file contains the tolerances used in the GPIB example controller
program
 * for the VM5000HD, main.c
 *
 * (c) Tektronix, Inc.
 *
 */

#ifndef TOLERANCE_H
#define TOLERANCE_H
// CONSTANTS FOR MEASUREMENT LIMIT CHECKING

  // Channel Delay
#define MAX_CHANNEL_DELAY   10.0F
#define MIN_CHANNEL_DELAY   -10.0F

  // Color Bars
#define NUMBER_OF_BARS 8
#define COLORBARS_TOLERANCE (50.0F)
const char* colorString[NUMBER_OF_BARS] = {
    "White",
    "Yellow",
    "Blue",
    "Green",
    "Magenta",
    "Red",
    "Blue",
    "Black",
  };
const float chan1Bar274mNominal[NUMBER_OF_BARS] = {
    700.0F, // White
    649.5F, // Yellow
    551.2F, // Blue
    500.6F, // Green
    199.4F, // Magenta
    148.8F, // Red
    50.5F,  // Blue
    0.0F,   // Black
    };
```

```
const float chan2Bar274mNominal[NUMBER_OF_BARS] = {
       0.0F, // White
    -350.0F, // Yellow
      80.0F, // Blue
    -269.8F, // Green
     269.8F, // Magenta
     -80.2F, // Red
     350.0F, // Blue
       0.0F, // Black
....};
const float chan3Bar274mNominal[NUMBER_OF_BARS] = {
       0.0F, // White
      32.0F, // Yellow
    -350.0F, // Blue
    -317.9F, // Green
     317.9F, // Magenta
     350.0F, // Red
     -32.0F, // Blue
       0.0F, // Black
};

// Multiburst
#define NOMINAL_MB_FLAG_AMPLITUDE  320.0F  // mV
#define NOMINAL_MB_FLAG_TOL        10.0F   // mV
#define MAX_MULTIBURST_MHZ_TOL   0.5F
#define MIN_MULTIBURST_MHZ_TOL  (-0.5F)
#define MAX_MULTIBURST_DB    2.0F
#define MIN_MULTIBURST_DB    (-2.0F)

#define NUMBER_OF_BURSTS 6
const float nominalBurstFrequencies[NUMBER_OF_BURSTS] = {
    5.0F,
    10.0F,
    15.0F,
    20.0F,
    25.0F,
    30.0F,
    };

  // Noise
#define MAX_NOISE_RMS_MV    10.0F
#define MIN_NOISE_RMS_MV    0.0F
```

```
// Nonlinearity
#define MAX_NONLINEARITY_PERCENT_DELTA  3.0F

  // Sync
#define MAX_NEG_SYNC_LEVEL  -250.0F
#define MIN_NEG_SYNC_LEVEL  -350.0F
#define MAX_POS_SYNC_LEVEL  350.0F
#define MIN_POS_SYNC_LEVEL  250.0F
#define MAX_FRONT_PORCH_LEVEL 20.0F
#define MIN_FRONT_PORCH_LEVEL -20.0F
#define MAX_FRONT_PORCH_WIDTH 727.27F
#define MIN_FRONT_PORCH_WIDTH 565.66F
#define MAX_NEG_SYNC_FALL   74.07F
#define MIN_NEG_SYNC_FALL   33.67F
#define MAX_NEG_SYNC_WIDTH  633.0F
#define MIN_NEG_SYNC_WIDTH  552.19F
#define MAX_SYNC_RISE       74.07F
#define MIN_SYNC_RISE       33.67F
#define MAX_POS_SYNC_WIDTH  633.0F
#define MIN_POS_SYNC_WIDTH  552.19F
#define MAX_POS_SYNC_FALL   74.07F
#define MIN_POS_SYNC_FALL   33.67F
#define MAX_BACK_PORCH_WIDTH  2114.5F
#define MIN_BACK_PORCH_WIDTH  1952.8F

#endif
```

Each of the six measurements has specific limits that define the maximum and minimum limit conditions that the measurement should be within. If these limits are exceeded then this will produce an error flag within the main program. The exception to this is the color bar measurement that requires a slightly different implementation because it requires three different parameters to be set for each of the channels and the tolerance is a deviation from the ideal value. In this case COLORBARS_TOLERANCE (50.0F) is defined and each color bar within each channel has a specific voltage value. This tolerances file is for one specific video standard and format, not all video formats will have the same specific conditions and it maybe necessary to have different limit files for each format being measured by the instrument.

There are several other include files that are set-up in the initial main.c program

```
/*
 * main.c version 1.0
 *
 * Example program to make measurements using the Tektronix
VM5000HD.
 * It is expected you will restructure this to fit your project's needs
 * changing the error reporting boundaries to fit your product.
 *
 * This is intended to provide an example only, helping you get started
 * Using the VM5000HD.
 *
 * This programs can be configured to some degree using the #define
below.
 *
 * (c) Tektronix, Inc.
 *
 */
#include <windows.h>
#include <stdio.h>
#include <string.h>

#include "decl-32.h"
#include "lib.h"
#include "tolerances.h"
```

The lib.h library is included because it allows the program to communicate with the GPIB system and send commands via the TDS oscilloscope platform on which the VM5000HD is based. Additional information on the GPIB capabilities and details of how to send commands to the instrument can be found in the following manual (071-0876-01) that is available on the VM5000HD install CD. There are basically two types of commands:

1. An instruction that commands the instrument to perform some parameter
2. A query, which asks a question as to the current state of the instrument, for instance whether it has completed a command. For example:

Command	
DISplay:GRAticuleFRAme	Changes the display graticule frame
Query	
ID?	Queries the instrument

NOTE: For TDS commands please follow the syntax within the manual.

The commands for the VM5000HD utilize a VARIable:VALue syntax. This mechanism signals the platform to send commands to the VM5000HD application. Full details on the instruction set and procedure can be found in Appendix C of the VM5000HD User Manual (071-1252-00). The following example illustrates how to set the color bar measurement on.

Command	
VARI:VAL "ColorBarsSet" , "on"	Sets color bar measurement on
Query	
VARI:VAL? "ColorBarsSet"	Queries the status of the color bar measurement on

NOTE: The commands used by the VM5000HD consist of the following instructions VARI:VAL Command, Argument where Command is cases sensitive and must use the correct upper and lower case character set to be understood by the system. The reason for this is that the value becomes a pointer internally within the VM5000HD application and the incorrect case will result in pointing to the wrong location. However Argument is NOT case sensitive because it is a value.

In order to ensure proper flow of your instructions within the VM5000HD it is required that a 50 millisecond delay is used to allow each command to propagate through the platform and the VM5000HD application. This prevents conflict arising within the system and is achieved by the following statements within the program.

```
// All times are in milliseconds
#define COMMAND_TIME_DELAY  50
// Use this delay between each command.
```

To simplify the query process and help determine when a command is completed a function QueryUntilReady is used. This function queries a command and loops until the command is finished, there are also several conditional statements to avoid a lock up on an error. The initial defines set the parameters for the QueryUntilReady function at the beginning of the program.

```
// These are values returned by QueryUntilReady(), to indicate the
result of the query
#define QUERYERROR (-1)
#define QUERYZERO  0
#define QUERYONE   1
#define QUERYOK    2
#define QUERYOTHER 3
```

The major function QueryUntilReady is defined later in the program and takes in char * which is used as a pointer to the command being queried. Two other values are input to the function. The int expectedResult is the value which is expected to be returned by the query when finished and int timeoutmilliSeconds is the maximum time allowed to wait to avoid a lock up condition.

```
/*
 * Function: Query the VM5000HD and wait for the desired response to
 * ensure the VM5000HD command processing is complete. This also has a
 * timeout to avoid locking up if an error occurs.
 *
 * Input Parameters:
 *  char *: pointing to the command being queried
 *  int: the expected result, see below
 *  timoutSeconds: maximum time to wait to avoid locking up
 *
 * Return Value: int indicating the value received from the VM5000HD
 * Returns the classification of query, as specified above
 *    QUERYERROR = error,
 *    QUERYZERO  = "Header 0",
 *    QUERYONE   = "Header 1",
 *    QUERYOK    = "Header OK",
 *    QUERYOTHER = all others
 */
int QueryUntilReady(const char *command, int expectedResult, int
timeoutmilliSeconds)
{
int milliSeconds = 0; // milliseconds
int returnVal = 0;

  // build the query to be sent to the VM5000HD: Variable:value?
  "command" \n
  strcpy(outString, "VARIable:VALue? \"");
  strcat(outString, command);
  strcat(outString, "\"\n");

  // get the VM5000HD response to the query
  returnVal = getVM5000HDResponse(outString);

  // wait until the response is what is expected or timeout occurs
  while ((milliSeconds < timeoutmilliSeconds) && (returnVal !=
  expectedResult))
  {
    returnVal = getVM5000HDResponse(outString);
    Sleep(10);
    milliSeconds = milliSeconds + 10;
  }

  // If a timeout occurred, notify the operator because there is probably
  an error.
if (milliSeconds >= timeoutmilliSeconds) {
  printf(" TIMEOUT in %s: waited > %d millliseconds\n", command,
timeoutmilliSeconds);
    returnVal = QUERYERROR;
  }
  return returnVal;
}
```

The following is the main part of the query statement which initiates the "VARIable:VALue? \ "" query. The following lines are the necessary concatenation of the strings to which the query is applied the following piece of the above function is shown below. This then equates to the following statement VARI:VAL? "ColorBarsSet" for example.

```
// build the query to be sent to the VM5000HD: Variable:value? "
command" \n
  strcpy(outString, "VARIable:VALue? \"");
  strcat(outString, command);
  strcat(outString, "\"\n");
```

There are several other define statements at the beginning of the program.

The PRINTFLAG can be used to display the GPIB commands of the VM5000HD. This allows debugging of the program if changes are made from the original program. Normally the value would be set to "0" so that these commands were not seen as the program runs. If the value is set to "1" then each command will be displayed as the program executes.

```
/* Use this define to display the GPIB commands to the VM5000HD
 * Set to 1 to see the commands; set to 0 to not see them.
 */
#define PRINTFLAG 0
```

The GPIB_ADDRESS sets the GPIB address of the platform. This is similar to the command available in the Utilities menu for the GPIB Configuration under address function.

```
/* Use this defines to set the GPIB address of your VM5000HD.
 * Set the address in the platform/oscilloscope:
 *       Utilities->GPIB Configuration menu under Address.
 */
#define GPIB_ADDRESS "dev1"
```

The program is written to execute each of the six measurements. If you wish not to execute one or several of the measurement then the define statement for that particular measurement should be commented out and then this function will not be tested as part of the measurement.

```
/* Use these defines to select which measurements you wish to run
 * If a define is commented out, that measurement is not run.
 */
#define MEASURE_CHANNELDELAY
#define MEASURE_COLORBARS
#define MEASURE_MULTIBURST
#define MEASURE_NONLINEARITY
#define MEASURE_NOISE
#define MEASURE_SYNC
```

Each of these define statements makes a call to a function doMeasurements in the program to perform the specific measurement. An initial setup is performed before executing the individual measurements. Note if these measurements are not defined in the first part of the program and are commented out then the ifdef (If Defined) command will not be performed on those specific measurements. This part of the program sets up the initial values of each measurement and initialize the loop counter which allows the measurements to be repeated a number of times. A tolerance value is setup to compare some of the measurements against their measurement limits and a check of the ID of the VM5000HD is performed.

```
/*
 * Function: Set up the VM5000HD to do all (or selected) measurements.
 *           select the desired measurements using defines at the top
 *           Read the result from the VM5000HD.
 *           Check the results are ok.
 *           Print a report of the resulting measurements.
 *
 *           Set this to make a single set of measurements or loop
continuously.
 *
 * Input Parameters: none
 *
 * Return Value: total number of errors
 */
int doMeasurements(void)
{
  // used for all measurements
float tolerence = 50.0F; // mV

#ifdef LOOP
  int loopCount = 0;
#endif /* LOOP */

#ifdef MEASURE_CHANNELDELAY
  float Ch1Delay,Ch2Delay,Ch3Delay;
#endif /* MEASURE_CHANNELDELAY */

#ifdef MEASURE_COLORBARS
  float Ch1Bar[8],Ch2Bar[8],Ch3Bar[8];
  long bar;
#endif /* MEASURE_COLORBARS */

#ifdef MEASURE_MULTIBURST /* turn ON the multiburst measurement */
  float Ch1Multiburst[NUMBER_OF_BURSTS];
  float Ch2Multiburst[NUMBER_OF_BURSTS];
  float Ch3Multiburst[NUMBER_OF_BURSTS];
  long burst;
#endif /* MEASURE_MULTIBURST */

#ifdef MEASURE_NOISE
  float Ch1Noise,Ch2Noise,Ch3Noise;
#endif /* MEASURE_NOISE */
```

```
#ifdef MEASURE_NONLINEARITY
#define NUMBER_OF_NONLINEARITY_DELTAS 5
  float Ch1Delta[NUMBER_OF_NONLINEARITY_DELTAS];
  float Ch2Delta[NUMBER_OF_NONLINEARITY_DELTAS];
  float Ch3Delta[NUMBER_OF_NONLINEARITY_DELTAS];
  long delta;
#endif /* MEASURE_NONLINEARITY */

#ifdef MEASURE_SYNC
 /* turn ON the sync measurement */
  float frontPorchLevel, negSyncLevel, posSyncLevel;
  float frontPorchWidth, negSyncFall, negSyncWidth, syncRise,
posSyncWidth, posSyncFall, backPorchWidth;
#endif /* MEASURE_SYNC */

int cdError, cbError, mbError, noiseError, nlError, syncError, totalErrors;
  cdError = cbError = mbError = noiseError = nlError = syncError =
totalErrors = 0;

// Send a query to the VM5000HD platform and get the response, the ID
  GpibWriteString ("id?", PRINTFLAG);
  GpibRead (queryResponse, MAX_BUF);
  if (PRINTFLAG == 1)
    printf ("%s", queryResponse);

  strcpy(outString, "VARIable:VALue? \"ID\"\n");
  GpibWriteString (outString, PRINTFLAG);
  GpibRead (queryResponse, MAX_BUF);
  if (PRINTFLAG == 1)
    printf ("Query response to id? %s\n", queryResponse);

#ifdef LOOP
  while (1)
  {
#endif /* LOOP */
    // Set up to run all measurements

    /* set the VM5000HD to its default setting to provide
     * a known starting setup
     */
    doDefaultSettings();
```

Note the measurement can be performed continuously if the define LOOP is not commented out of the program otherwise the program will only run through the measurements once.

```
/* Set this define "on" to cause the measurements to loop continuously
 * If this is commented out, the measurements run only once.
 */
#define LOOP
```

Once these initial settings are performed the command for the measurements are made. In this case the "VARIable:VALue ChannelDelaySet","on" command is issued for each of the measurement which are defined in the first part of the program. Note the default of the instrument is to have color bar measurement selected. Therefore if the color bars measurement is commented out in the first part of the program it is necessary to de-select the color bar measurement and this is done by the ifndef (If Not Defined) and the command to turn off the measurement is issued instead of turning on the measurement as is done with all the other commands.

```
#ifdef MEASURE_CHANNELDELAY /* turn ON the channel delay
    measurement */
    strcpy(outString, "VARIable:VALue \"ChannelDelaySet\",\"on\"\n");
    sendCommandToVM5000HD (outString);
#endif

  /* note this turns off Color Bars; Color Bars is turned on by default
*/
#ifndef MEASURE_COLORBARS /* turn OFF the channel delay
....measurement */
    strcpy(outString, "VARIable:VALue \"ColorBarsSet\",\"off\"\n");
    sendCommandToVM5000HD (outString);
#endif

#ifdef MEASURE_MULTIBURST /* turn ON the multiburst measurement */
    strcpy(outString, "VARIable:VALue \"MultiburstSet\",\"on\"\n");
    sendCommandToVM5000HD (outString);
#endif

#ifdef MEASURE_NOISE /* turn ON the Noise measurement */
    strcpy(outString, "VARIable:VALue \"NoiseSet\",\"on\"\n");
    sendCommandToVM5000HD (outString);
#endif

#ifdef MEASURE_NONLINEARITY /* turn ON the NonLinearity
    measurement */
    strcpy(outString, "VARIable:VALue \"NonLinearitySet\",\"on\"\n");
    sendCommandToVM5000HD (outString);
#endif

#ifdef MEASURE_SYNC /* turn ON the sync measurement */
    strcpy(outString, "VARIable:VALue \"SyncSet\",\"on\"\n");
    sendCommandToVM5000HD (outString);
#endif
```

After the measurements have been completed the set of results need to be obtained and compared with the tolerances data. Then the results can be reported as an output and provide indication to the user of a pass or failure of the DUT. So each of the measurements scans the data values of the VM5000HD and then by performing the GpibRead vm5000hdResult. It then compares these values against the maximum and minimum values of the tolerances.h file and if the value is outside of these limits an error is reported. Note if there are no errors then no values are printed as a report and the program indicates that the device passed the series of tests.

```
#ifdef MEASURE_CHANNELDELAY
    // Channel Delay Results
    strcpy(outString, "VARIable:VALue? \"ChannelDelayAll\"\n");
    GpibWriteString (outString, PRINTFLAG);
    GpibRead (vm5000hdResult, MAX_BUF);
    if (1 == PRINTFLAG)
      printf ("%s",vm5000hdResult);

    sscanf(vm5000hdResult,"%s%f%f%f",dummyString, &Ch1Delay,
&Ch2Delay, &Ch3Delay );

    if (Ch1Delay > MAX_CHANNEL_DELAY || Ch1Delay
< MIN_CHANNEL_DELAY)
    {
      cdError++;
      printf ("String read %s\n",vm5000hdResult);
      printf ("Ch1 ChannelDelay = %f\n", Ch1Delay);
    }

    if (Ch2Delay > MAX_CHANNEL_DELAY || Ch2Delay
< MIN_CHANNEL_DELAY)
    {
      cdError++;
      printf ("String read %s\n",vm5000hdResult);
      printf ("Ch2 ChannelDelay = %f\n", Ch2Delay);
    }

    if (Ch3Delay > MAX_CHANNEL_DELAY || Ch3Delay
< MIN_CHANNEL_DELAY)
    {
      cdError++;
      printf ("String read %s\n",vm5000hdResult);
   printf ("Ch3 ChannelDelay = %f\n", Ch2Delay);
```

```
#ifdef MEASURE_COLORBARS
    // Color Bars Results
    tolerence = COLORBARS_TOLERANCE; // mV

    strcpy(outString, "VARIable:VALue? \"ColorBarsmVCh1\"\n");
    GpibWriteString (outString, PRINTFLAG);
    GpibRead (vm5000hdResult, MAX_BUF);
    if (1 == PRINTFLAG)
      printf ("%s",vm5000hdResult);

    sscanf(vm5000hdResult,"%s%f%f%f%f%f%f%f%f",dummyString,
&Ch1Bar[0],&Ch1Bar[1],&Ch1Bar[2],&Ch1Bar[3],&Ch1Bar[4],
&Ch1Bar[5],&Ch1Bar[6],&Ch1Bar[7]);
    for(bar = 0; bar < NUMBER_OF_BARS; bar++)
      if( (Ch1Bar[bar] > chan1Bar247mNominal[bar]+tolerence) ||
(Ch1Bar[bar] < chan1Bar247mNominal[bar]-tolerence) )
      {
        cbError++;
        printf ("ColorBars Ch1 %s = %f vs nominal of
%f\n",&colorString[bar],Ch1Bar[bar], chan1Bar247mNominal[bar]);
        printf ("String read %s\n",vm5000hdResult);
      }
    strcpy(outString, "VARIable:VALue? \"ColorBarsmVCh2\"\n");
    GpibWriteString (outString, PRINTFLAG);
    GpibRead (vm5000hdResult, MAX_BUF);
    if (1 == PRINTFLAG)
      printf ("%s",vm5000hdResult);

    sscanf(vm5000hdResult,"%s%f%f%f%f%f%f%f%f",dummyString,
&Ch2Bar[0],&Ch2Bar[1],&Ch2Bar[2],&Ch2Bar[3],&Ch2Bar[4],
&Ch2Bar[5],&Ch2Bar[6],&Ch2Bar[7]);
    for(bar = 0; bar < NUMBER_OF_BARS; bar++)
      if( (Ch2Bar[bar] > chan2Bar247mNominal[bar]+tolerence) ||
(Ch2Bar[bar] < chan2Bar247mNominal[bar]-tolerence) )
      {
        cbError++;
        printf ("ColorBars Ch2 %s = %f vs nominal of
%f\n",&colorString[bar],Ch2Bar[bar], chan2Bar247mNominal[bar]);
        printf ("String read %s\n",vm5000hdResult);
      }
    strcpy(outString, "VARIable:VALue? \"ColorBarsmVCh3\"\n");
    GpibWriteString (outString, PRINTFLAG);
    GpibRead (vm5000hdResult, MAX_BUF);
    if (1 == PRINTFLAG)
      printf ("%s",vm5000hdResult);

    sscanf(vm5000hdResult,"%s%f%f%f%f%f%f%f%f",dummyString,
&Ch3Bar[0],&Ch3Bar[1],&Ch3Bar[2],&Ch3Bar[3],&Ch3Bar[4],
&Ch3Bar[5],&Ch3Bar[6],&Ch3Bar[7]);
    for(bar = 0; bar < NUMBER_OF_BARS; bar++)
      if( (Ch3Bar[bar] > chan3Bar247mNominal[bar]+tolerence) ||
(Ch3Bar[bar] < chan3Bar247mNominal[bar]-tolerence) )
      {
        cbError++;
        printf ("ColorBars Ch3 %s = %f vs nominal of
%f\n",&colorString[bar],Ch3Bar[bar], chan3Bar247mNominal[bar]);
        printf ("String read %s\n",vm5000hdResult);
      }
#endif /* MEASURE_COLORBARS */
```

Channel Delay is a relatively simple procedure to query the data and check that it is within limits. A similar function is done for Color Bars but is a little more complicated because of the greater number of values reported by the VM5000HD and the larger number of limits to be checked for each individual result. Therefore an array is used to more effective pass the results and limits data from the VM5000HD to the program.

Once the measurements have been completed it is useful to generate a report of the results to document as to whether the DUT passed or failed the series of tests and to provide a log of measurement results from a set of devices. This is achieved by the function GENERATE_REPORT, which saves the results to a file.

```
/* Use this defines to generate a report when the tests complete
 * If the define is commented out, no report is generated.
 */
#define GENERATE_REPORT
```

If the define GENERATE_REPORT is not commented out of the program then the following function is performed at the completion of the measurements. If the define for GENERATE_REPORT is commented out of the program then no report is generated.

```
#ifdef GENERATE_REPORT
    // Generate report of every measurement

    strcpy(outString, "VARIable:VALue
 \"ReportGenerate\",\"c:\\VM5000HD\\FullTest.rtf\"\n");
    generateReport (outString);
#endif /* GENERATE_REPORT */

    totalErrors = cdError + cbError + mbError + noiseError + nlError +
syncError;
    if (cdError > 0 || cbError > 0 || mbError > 0 || noiseError > 0 ||
nlError > 0 || syncError > 0)
    {
      printf ("Total Errors = %d \n",totalErrors);
      printf ("cdError = %d, cbError = %d, mbError = %d, noiseError =
%d, nlError = %d, syncError = %d\n",
        cdError, cbError, mbError, noiseError, nlError, syncError);
    }
#ifdef LOOP
    loopCount++;

    if (totalErrors == 0 )
    {
      printf ("Unit passed: Loop count = %d\n", loopCount);
    }
    else
    {
      printf ("Unit failed: Total errors = %d\n",totalErrors);
    }
  }
#endif /* LOOP */
  return totalErrors;
}
```

The initial VARIable:VALue "ReportGenerate","c:\\VM5000HD\\FullTest.rtf" statement saves the results from the measurements in a file stored on the VM5000HD under the directory c:\VM5000HD\ and the file is named FullTest.rtf. The data from the file can be imported into a spreadsheet application for further analysis. In a manufacturing environment it maybe useful for the operator to input the serial number of the device under test and this value could be used for the name of the file. Then information from each device could be correlated in a spreadsheet to determine various parameters of the DUT and identify if any possible problems exist on the production line.

The function generateReport is called within the above part of the program. This part of the program performs the report command to the VM5000HD and then queries the instrument until the report is ready and passes the value back to the other part of the program above.

```
/*
 * Function: send the generateReport (which includes the report
path name)
 *           command to the VM5000HD and wait for it to complete.

 *           When complete the VM5000HD will return "OK" to the query.
*
 * Input Parameters: full command for the VM5000HD including the
report path and type
 *
 * Return Value: none
 */
void generateReport (char * reportCommandWithPath)
{
int status;
  // Send the report command to the VM5000HD and wait for it to complete
  status = GpibWriteString (reportCommandWithPath, PRINTFLAG);
  Sleep(COMMAND_TIME_DELAY);

  // ReportGenerate command: Wait until ReportGenerate has finished
  QueryUntilReady("ReportGenerate", QUERYOK,
  REPORTGENERATE_TIMEOUT);

  // Check for Errors and Warnings
  ErrorOrWarningCheck("Error", "ReportGenerate");
  ErrorOrWarningCheck("Warning", "ReportGenerate");
  return;
}
```

The main body of the program is the following series of commands

```
/* starting point for the C code is main () */
void main (argc, argv)
    int     argc;
    char    *argv[];
{
  int totalErrors = 0;

  strcpy (openDevice, GPIB_ADDRESS);/* set the address of the VM5000HD
*/
  GpibOpen (openDevice);            /* open the VM5000HD */

#ifdef HOSTSTART
  // start the VM5000HD application, but wait until it's started
  GpibWriteString ("APPLICATION:ACTIVATE \"VM5000HD\"", PRINTFLAG);
  Sleep (15000);
#endif /* HOSTSTART */

  totalErrors = doMeasurements();
  if (totalErrors == 0 )
  {
    printf ("Unit passed, no errors\n");
  }
  else
  {
    printf ("Unit failed: Total errors = %d\n",totalErrors);
  }

#ifdef HOSTSTART
  strcpy(outString, "VARIable:VALue \"application\",\"exit\"\n");
  sendCommandToVM5000HD (outString);
#endif /* HOSTSTART */

  GpibClose ();
}
```

The program starts by setting the address of the VM5000HD. Then the GPIB communication is opened between the program and the VM5000HD. After communication is open the message APPLICATION:ACTIVATE "VM5000HD" is sent to the platform to start the VM5000HD measurement software. There is a 15 second sleep allowing the application software to start.

```
// use this define, HOSTSTART, if you want the host controller to start
// the VM5000HD measurement software.  Otherwise start it manually
// in the menu: File->RunApplication->VM5000HD
//#define HOSTSTART
```

The #define HOSTSTART allows your software program to start the VM5000HD application as if the user had manually started the application from the menu File->RunApplication –>VM5000HD. Typically this function is commented out of the program and the application must already be running on the VM5000HD.

It is useful to start the VM5000HD in a known state so that the instrument is configured correctly before the measurements are started. This is achieved by several functions, the first of these DefaultSettings restore the instrument to its factory default settings.

```
/*
 * Function: send the DefaultSettings command to the VM5000HD and
wait for it to
 *           complete.  When complete the VM5000HD will return "OK"
to the query.
 *
 * Input Parameters: none
 *
 * Return Value: none
 */
void doDefaultSettings(void)
{
int status;
  // Send the execute command to the VM5000HD and wait for it to
complete
  strcpy(outString, "VARIable:VALue \"DefaultSettings\",\"1\"\n");
  status = GpibWriteString (outString, PRINTFLAG);
  Sleep(COMMAND_TIME_DELAY);

  // DefaultSettings command: Wait until DefaultSettings has finished
  QueryUntilReady("DefaultSettings", QUERYOK,
RECALL_SAVE_DEFAULT_TIMEOUT);

  // Check for Errors and Warnings
  ErrorOrWarningCheck("Error", "DefaultSettings");
  ErrorOrWarningCheck("Warning", "DefaultSettings");
  return;
}
```

This function issues the command "VARIable:VALue "Default Settings","1" which restore the VM5000HD software to its default settings and then waits for 50ms for the command to be passed through to the VM5000HD application and then queries the instrument using the QueryUntilReady function until the VM5000HD has restore the default settings or until a timeout occurs.

You may wish to configure the VM5000HD using your own particular settings rather than the factory defaults. This can be achieved by RecallSettings function that takes the file name and the path of the specific file you wish to recall and then queries the instrument using the function QueryUntilReady until a value of "OK" is returned indication completion of the command, if not the function may timeout or produce error warnings. This program uses the DefaultSettings command and does not use RecallSettings, however the function is within the main program for developers who wish to create their own specific settings.

```
/*
 * Function: send the recallSettings (which includes the settings file
 *path name)command to the VM5000HD and wait for it to complete.
 * When complete the VM5000HD will return "OK" to the query.
 *
 * Input Parameters: full command for the VM5000HD including the
 * settings file path
 * Return Value: none
 */
void recallSettings (char * recallCommandWithPath)
{
int status;
  // Send the report command to the VM5000HD and wait for it to complete
  status = GpibWriteString (recallCommandWithPath, PRINTFLAG);
  Sleep(COMMAND_TIME_DELAY);

  // RecallSettings command: Wait until RecallSettings has finished
  QueryUntilReady("RecallSettings", QUERYOK,
RECALL_SAVE_DEFAULT_TIMEOUT);

  // Check for Errors and Warnings
  ErrorOrWarningCheck("Error", "RecallSettings");
  ErrorOrWarningCheck("Warning", "RecallSettings");
  return;
}
```

You may wish in some configurations to actually save the settings of the instrument within the programs. A function called SaveSettings is available for this procedure. This function takes the file name and the path of the specific file you wish to save and then queries the instrument using the function QueryUntilReady until a value of "OK" is returned indication completion of the command, if not the function may timeout or produce error warnings. Saved settings are assigned the extension .vmset.

```
/*
 * Function: send the saveSettings (which includes the settings file
path name)
 *           command to the VM5000HD and wait for it to complete.
 *           When complete the VM5000HD will return "OK" to the query.
 *
 * Input Parameters: full command for the VM5000HD including the
settings file path
 *
 * Return Value: none
 */
void saveSettings (char * saveCommandWithPath)
{
int status;
  // Send the report command to the VM5000HD and wait for it to complete
  status = GpibWriteString (saveCommandWithPath, PRINTFLAG);
  Sleep(COMMAND_TIME_DELAY);

  // SaveSettings command: Wait until SaveSettings has finished
  QueryUntilReady("SaveSettings", QUERYOK,
  RECALL_SAVE_DEFAULT_TIMEOUT);

  // Check for Errors and Warnings
  ErrorOrWarningCheck("Error", "SaveSettings");
  ErrorOrWarningCheck("Warning", "SaveSettings");
  return;
}
```

To prevent lockup each function has a condition for timeout of a command. This time is set up by an initial set of values at the start of the program. After the time interval specified the program will have waited for sufficient time for the command to execute and there has still been no response from the instrument, therefore the command will timeout and prevent the program from locking up.

```
#define RESET_ERROR_OR_WARNING_DELAY 500     // This delay is
used when resetting Error or Warning

// These timeouts are "safety nets", so the code will not hang if there
is an unexpected problem
// They are passed into QueryUntilReady(), where the timeout check is
performed
// If the timeouts are exceeded, QueryUntilReady() will print an error
message
#define EXECUTE_TIMEOUT                150000
#define ERROR_AND_WARNING_TIMEOUT     5000
#define REPORTGENERATE_TIMEOUT        5000
#define RECALL_SAVE_DEFAULT_TIMEOUT  5000
```

```
/*
 * Function: Query the VM5000HD for Errors or Warnings
 * If a error/warning has been generated, it is cleared by
 * writing the string "Off".  This does not affect the error/warning
 * on/off reporting control.  That command is "ErrorReporting".
 *
 * Input Parameters:
 *  char *: the string "Error" or "Warning", which is to be checked
 *  char *: The current command that was sent to the VM5000HD
 *
 * Return Value: 0 for no error nor warning
 *
 * Note: the actual warning (the string returned) is left in
* queryResponse buffer so the calling program can analyze it if
* needed.
 */
int ErrorOrWarningCheck(const char *errorOrWarning, char
*currentCommand)
{
int resetErrorWarning = 0;
int returnVal;
char *pdest;

  strcpy(errorWarningCheck, "VARIable:VALue? \"");
  strcat(errorWarningCheck, errorOrWarning);
  strcat(errorWarningCheck, "\"\n");

  returnVal = getVM5000HDResponse(errorWarningCheck);

  if ( returnVal != QUERYZERO) {
    /* If command sent was generated an Error or Warning then we
 should wait a
     * while to see if it changes to "0" before writing out the command
again.
     */
    resetErrorWarning = TRUE;
    pdest = strstr( currentCommand, errorOrWarning );
    if( pdest != NULL ) { // this is resetting Error or Warning to OFF
      // Sleep, re-read it, if it's now 0, no need to reset it.
      Sleep(RESET_ERROR_OR_WARNING_DELAY);
      if (QUERYZERO == getVM5000HDResponse(errorWarningCheck)) {
        resetErrorWarning = FALSE;
      }

    }
    if (TRUE == resetErrorWarning) {
      strcpy(errorWarningCheck, "VARIable:VALue \"");
      strcat(errorWarningCheck, errorOrWarning);
      strcat(errorWarningCheck, "\",\"off\"\n");
      GpibWriteString (errorWarningCheck, PRINTFLAG);
      // Wait until "Error/Warning off" has been cleared before doing
the next command
      QueryUntilReady(errorOrWarning, QUERYZERO, ERROR_AND_
WARNING_TIMEOUT);
    }
  }
  return returnVal;
}
```

In other cases the VM5000HD produces a variety of errors or warning messages and the program can also report these conditions. The function ErrorOrWarningCheck provides this facility. The inputs to the function are the Error or Warning, which is to be checked and the current command that the VM5000HD has performed. The query is issued to check whether an error or warning is present. If an error is present a sleep command is issued to wait and see if the error or warning is cleared. If not then the error or warning is set to "0" and a check is performed to see if it has cleared before executing the next command.

There are several functions within the program which are provided as a means to send command to the VM5000HD and to check for errors and warnings from the instrument. The function sendCommandToVM5000HD takes input as a char* string which is the command and returns an integer value to indicate whether there was an error or warning generated by the command. The remote GPIB command is sent by the command GpibWriteString (stringToVM5000HD, PRINTFLAG). To allow this command to be passed to the VM5000HD application a sleep command of 50 millsecond is performed. Finally a check for errors or warnings is done to ensure the command was received correctly and the correct action was done by the VM5000HD. The value of errorStatus and warningStatus are returned from the function.

```
/*
 * Function: Send a string to the VM5000HD and
 * sleep while the command propagates through the VM5000HD SW
 *
 * Input Parameters:
 * string: a char * pointing to the string to write to the VM5000HD
 *
 * Return Value: integer returned from ErrorOrWarningCheck to indicate
 * if there was an error or warning generated by this command
 */
int sendCommandToVM5000HD(char * stringToVM5000HD)
{
int status;
int errorStatus = 0;
int warningStatus = 0;

  /* Send the Remote Command */
  status = GpibWriteString (stringToVM5000HD, PRINTFLAG);
  Sleep(COMMAND_TIME_DELAY);

  // Check for Errors and Warnings
  errorStatus = ErrorOrWarningCheck("Error", stringToVM5000HD);
  warningStatus = ErrorOrWarningCheck("Warning", stringToVM5000HD);

  return (errorStatus + warningStatus);
}
```

The next function getVM5000HDResponse is used when specific response is required back from the VM5000HD application and this function checks for that response. The function takes the char*stringToVM5000HD input pointing to the string written to the VM5000HD and returns a value that corresponds to a specific query classification. The GPIB command is sent to the VM5000HD application GpibWriteString (stringToVM5000HD, PRINTFLAG) and then waits for the response from the application. The GPIB command GpibRead (queryResponse, MAX_BUF) reads the response back from the VM5000HD and truncates the string to the major component of the answer. This value is then checked and compared to the query classification. The value of the query results is then returned by the function.

```
/*
 * Function: Some commands provide a specific response when they are
 * done.
 * This function checks for that response.
 *
 * Input Parameters:
 * string: a char * pointing to the string to write to the VM5000HD
 *
 * Return Value: int indicating the value received from the VM5000HD
 * Returns the classification of query, as specified by
 * queryReturnType in lib.h
 *    QUERYERROR = error,
 *    QUERYZERO  = "Header 0",
 *    QUERYONE   = "Header 1",
 *    QUERYOK    = "Header OK",
 *    QUERYOTHER = all others
 */

int getVM5000HDResponse(char *stringToVM5000HD)
{
char stringToAnalyze [MAX_BUF + 1];

  // valued returned by the Gpib functions int status;

  // The result of the query from the VM5000HD that is returned to the
caller.
int queryResult = QUERYERROR;

  // This pointer will be set to point to the response past the header
char *strWithoutHeader;
unsigned int indexIntoString;

  /* write the query to the VM5000HD
   */
  status = GpibWriteString (stringToVM5000HD, PRINTFLAG);
        if ( status < 0)
        {
    printf("GPIB write error!!!!!\n");
                    GpibError ("Write Error:");
                    return queryResult;
        }

  /* read the response from the VM5000HD into queryResponse
   */
        queryResponse[0] = '\0';
  status = GpibRead (queryResponse, MAX_BUF);

  if ( status < 0)
  {
    printf("GPIB read error!!!!!\n");
    GpibError ("Read Error:");
    return queryResult;
  }

  // copy the string for analysis so the query response is still available
  strcpy (stringToAnalyze,queryResponse);

  // Remove the \n at the end of the response
  while (stringToAnalyze[strlen(stringToAnalyze)-1] == '\n') {
      stringToAnalyze[strlen(stringToAnalyze)-1] = '\0';
  }
  // remove the " mark
  while (stringToAnalyze[strlen(stringToAnalyze)-1] == '\"') {
      stringToAnalyze[strlen(stringToAnalyze)-1] = '\0';
  }
  while (stringToAnalyze[0] == '\"') {
      for (indexIntoString = 0; indexIntoString <
strlen(stringToAnalyze); indexIntoString++)
        stringToAnalyze[indexIntoString] =
stringToAnalyze[indexIntoString+1];
  }
  // remove the header from the response (all char up to the space)
  strWithoutHeader = strchr(stringToAnalyze, ' ');
  if (strWithoutHeader != 0) {
      strWithoutHeader = strWithoutHeader+1; // skip space
  }
  if (strWithoutHeader == 0) {
      queryResult = QUERYOTHER;
  }
  // see if the response is a zero (execute returns a 1 or 0)
  else if (strcmp(strWithoutHeader, "0") == 0) {
      queryResult = QUERYZERO;
  } // see if the response is a one (execute returns a 1 or 0)
  else if (strcmp(strWithoutHeader, "1") == 0) {
      queryResult = QUERYONE;
  }
  // see if the response is a OK
  // Save settings, Recall settings, Default settings
  // and Generate Report all return "OK" when done.
  else if (strcmp(strWithoutHeader, "OK") == 0) {
      queryResult = QUERYOK;
  }
  else { // unknown string
      queryResult = QUERYOTHER;
  }
  return queryResult;
}
```

The function doExecute provide a means to send execute commands to the VM5000HD application and then waits until the instrument has performed the operation using the QueryUntilReady function. When the operation is complete a value of "0" will be returned by the function. If the command is not executed correctly then the command will timeout or the program will flag an error or warning.

```
/*
* Function: send the Execute command to the VM5000HD and wait for the
* execute to complete.  When complete the VM5000HD will return "0" to
* the query.
* If the VM5000HD "RunMode" is set to "continuous", this will
* time out to avoid locking up if an error occurs.
*
* Input Parameters: none
*
* Return Value: none
*/
void doExecute(void)
{
int status;
  // Send the execute command to the VM5000HD and wait for it to
complete
  strcpy(outString, "VARIable:VALue \"Execute\",\"1\"\n");
  status = GpibWriteString (outString, PRINTFLAG);

    // Execute command: Wait until Execute has finished
  QueryUntilReady("Execute", QUERYZERO, EXECUTE_TIMEOUT);

  // Check for Errors and Warnings
  ErrorOrWarningCheck("Error", "Execute");
  ErrorOrWarningCheck("Warning", "Execute");
  return;
}
```

These various functions are used within various parts of the program and in some cases make calls to other function to check various parameters. To show an example of the output from the program an color bar measurement was made, with all other measurements commented out and the PRINTFLAG was set "1" to show the execution of commands (Note not all commands are show in the example because these were repeated commands of queries to the instrument which had not finished its instructions).

```
\GPIBExamples\Release>gpibexamples.exe
...writing id?
ID TEK/TDS5104,CF:91.1CT,FV:1.1.165+
...writing VARIable:VALue? "ID"

Query response to id? "ID Tek/VM5000HD FV:1.0"

...writing VARIable:VALue "DefaultSettings","1"

...writing VARIable:VALue? "DefaultSettings"

...writing VARIable:VALue? "Error"

...writing VARIable:VALue? "Warning"

...writing VARIable:VALue "RunMode","Once"

...writing VARIable:VALue? "Error"

...writing VARIable:VALue? "Warning"

...writing VARIable:VALue "Execute","1"

...writing VARIable:VALue? "Execute"

...writing VARIable:VALue? "Error"

...writing VARIable:VALue? "Warning"

...writing VARIable:VALue? "ColorBarsmVCh1"

"ColorBarsmVCh1 696.56 648.56 550.70 499.76 200.01 148.26
50.44 -0.53"
...writing VARIable:VALue? "ColorBarsmVCh2"

"ColorBarsmVCh2 -0.09 -349.82 78.66 -268.13 267.12 -78.78
346.11 0.45"
...writing VARIable:VALue? "ColorBarsmVCh3"

"ColorBarsmVCh3 0.24 31.93 -348.64 -316.45 313.13
345.99 -31.59 0.13"
...writing VARIable:VALue "ReportGenerate","c:\VM5000HD\FullTest.rtf"

...writing VARIable:VALue? "ReportGenerate"

...writing VARIable:VALue? "Error"

...writing VARIable:VALue? "Warning"

...writing VARIable:VALue "Warning","off"

...writing VARIable:VALue? "Warning"

Unit passed, no errors
```

The time it takes to perform each of these measurements can be a critical factor in a manufacturing environment. The ability to shave seconds off the manufacturing time can have drastic performance improvements for the production line. The following table shows the time it takes to perform each measurement individually and the time it takes to completely perform all measurements once. These values are typical results and could depend on the speed of the computer used to run this test program.

	Channel Delay (Sec)	Color Bars (Sec)	Multi-burst (Sec)	Non-Linearity (Sec)	Noise (Sec)	Sync (Sec)	Total Measurement Time
AutoScale ON	2.11	1.92	1.84	12.06	1.84	1.75	28.59
AutoScale OFF	4.03	1.94	2.06	14.98	6.39	1.94	35.59

Conclusion

The purpose of this program is to provide an outline of how to control the VM5000HD application and how to issue commands and queries to the instrument. This will allow the VM5000HD analog HD component measurements to be incorporated into automation routine and allow simple pass/fail indication of the device under test. This program can be adapted for your own specific application and provides a starting point from which to develop your own manufacturing routine. Parts or this entire program can be cut and pasted into your own routines and developed for your own application. However Tektronix is not responsible for support of this program and its derivatives.

VM5000HD Automated Video Measurement Set

The VM5000HD automates a variety of component analog video measurements utilized to verify the integrity and quality of HDTV video signals. It automatically assesses conformance of selected video signal parameters to applicable EIA-770-3, SMPTE-274M and 296M standards. It also automates measurement of other industry-standard video parameters used to quantify the performance of digital set-top boxes or other consumer video reception and play-out devices with component analog interfaces.

Overview of VM5000HD Features:

- Fast, accurate and repeatable video measurements
- Fully automated, comprehensive component analog video measurements
- Supports HDTV, progressive scan and PC format signals (Y'P'bP'r and RGB)
- Acquisition bandwidth and high sample rates for HDTV signals
- Extensive documentation capabilities
- Standard GPIB and LAN remote control capability

Contact Tektronix:

ASEAN / Australasia / Pakistan (65) 6356 3900
Austria +43 2236 8092 262
Belgium +32 (2) 715 89 70
Brazil & South America 55 (11) 3741-8360
Canada 1 (800) 661-5625
Central Europe & Greece +43 2236 8092 301
Denmark +45 44 850 700
Finland +358 (9) 4783 400
France & North Africa +33 (0) 1 69 86 80 34
Germany +49 (221) 94 77 400
Hong Kong (852) 2585-6688
India (91) 80-2275577
Italy +39 (02) 25086 1
Japan 81 (3) 3448-3010
Mexico, Central America & Caribbean 52 (55) 56666-333
The Netherlands +31 (0) 23 569 5555
Norway +47 22 07 07 00
People's Republic of China 86 (10) 6235 1230
Poland +48 (0) 22 521 53 40
Republic of Korea 82 (2) 528-5299
Russia, CIS & The Baltics +358 (9) 4783 400
South Africa +27 11 254 8360
Spain +34 (91) 372 6055
Sweden +46 8 477 6503/4
Taiwan 886 (2) 2722-9622
United Kingdom & Eire +44 (0) 1344 392400
USA 1 (800) 426-2200
USA (Export Sales) 1 (503) 627-1916
For other areas contact Tektronix, Inc. at: 1 (503) 627-7111

Updated 20 September 2002

For Further Information

Tektronix maintains a comprehensive, constantly expanding collection of application notes, technical briefs and other resources to help engineers working on the cutting edge of technology. Please visit **www.tektronix.com**

09/03 FL5639/WWW 25W-16961-0

Tektronix
Enabling Innovation