# TLA700 Series Logic Analyzers Remote Control with Support for Advanced Data Analysis

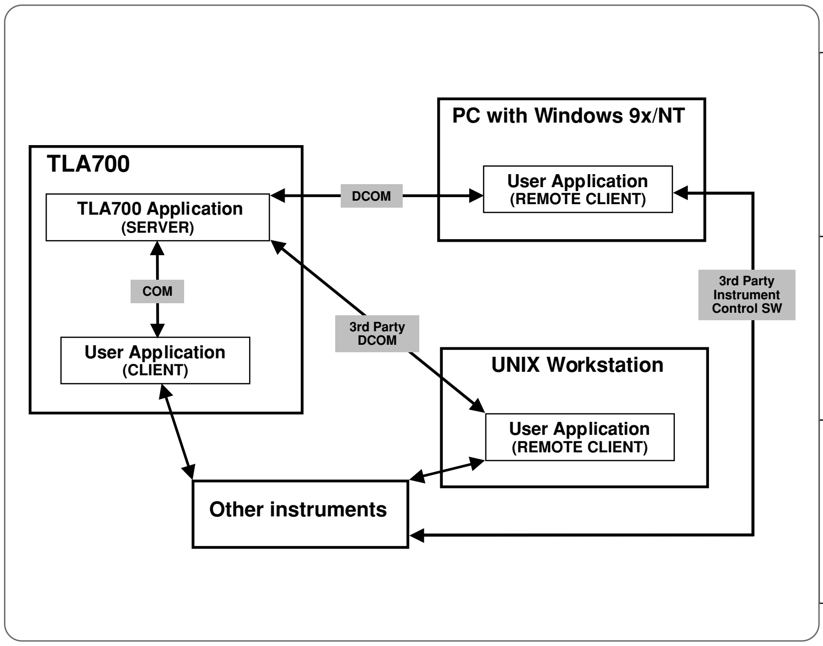

PC with Windows 9x/NT
User Application (REMOTE CLIENT)
TLA700
TLA700 Application (SERVER)
DCOM
COM
User Application (CLIENT)
3rd Party DCOM
3rd Party Instrument Control SW
UNIX Workstation
User Application (REMOTE CLIENT)
Other instruments

***Figure 1.*** *Block Diagram of TLA700 Programmatic Interface (TPI).*

Logic analyzers are used primarily as a debug tool for the digital design team. This team encompasses hardware developers, hardware/software integrators, and software developers. While the primary interface with the logic analyzer is via the display and keyboard/mouse, there are times when developers need to remotely operate the logic analyzer from a computer as a data acquisition unit.

Other times, they need to perform more advanced analysis upon the acquired data beyond that provided by the logic analyzer. To do this, they need the ability to easily move data from the logic analyzer to computer-based applications so they can customize the data display in order to analyze the data in a custom format (e.g., performance profiling of deep real-time trace, FFT analysis, etc.).

Applications for this capability can be found in engineering development (debug, verification, validation, characterization, etc.), as well as manufacturing (test, quality assurance, and failure analysis).

The TLA700's integrated PC, running Windows® 95, provides both remote control operation and support for advanced data analysis through the TLA700 Programmatic Interface (TPI).

## TPI Overview

TPI, which comes standard on the TLA700 logic analyzer, is based upon Microsoft's Component Object Module (COM) technology. This provides inter-program operability between the TLA700 application and other user applications (see Figure 1). When two objects communicate using COM/DCOM, one is the client and the other is the server. With TPI, the TLA700 is always the server and the user application is the client. You can control the TLA700 from a separate user program running either locally on the TLA700 (client) or over the network on an associated PC running Windows 9x/NT (remote client) via LAN using DCOM (Distributed COM). UNIX environments are supported via DCOM-on-UNIX software available from third parties such as Microsoft:

`www.microsoft.com/oledev`

Or Software AG:

`www.sagus.com`

For further details on COM/DCOM, refer to Microsoft's Web site for COM/DCOM:

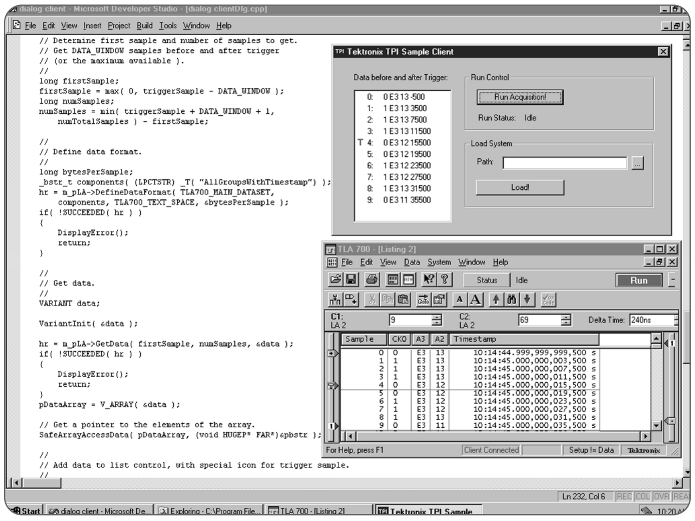
`www.microsoft.com/oledev`

The TLA application software exports four types of objects to the user's client program as shown in the object hierarchy in Figure 2.

Each object has an interface, composed of a set of methods (or functions) that can be called by the client program. Your client program can create an Application object to initially connect to the TLA700 application and to subsequently obtain a reference to a System object. The Application object exports a single interface called IApplication.

The System object provides methods for configuration, acquisition control, and save/load operations. Every client program must obtain a reference to a System object before it can obtain references to module objects. The System object exports a single interface called ISystem.

Tektronix
Enabling Innovation


dialog client - Microsoft Developer Studio - [dialog clientDlg.cpp]
File Edit View Insert Project Build Tools Window Help
Tektronix TPI Sample Client
Run Acquisition!
Load!
TLA 700 - [Listing 2]
File Edit View Data System Window Help
Status
Run
For Help, press F1
Client Connected
Setup != Data
Tektronix
Ln 232, Col 6
Start
dialog client - Microsoft De...
Exploring - C:\Program File...
TLA 700 - [Listing 2]
Tektronix TPI Sample
10:20 A

```
    // Determine first sample and number of samples to get.
    // Get DATA_WINDOW samples before and after trigger
    // (or the maximum available ).
    //
    long firstSample;
    firstSample = max( 0, triggerSample - DATA_WINDOW );
    long numSamples;
    numSamples = min( triggerSample + DATA_WINDOW + 1,
        numTotalSamples ) - firstSample;

    //
    // Define data format.
    //
    long bytesPerSample;
    _bstr_t components( (LPCTSTR) _T( "AllGroupsWithTimestamp") );
    hr = m_pLA->DefineDataFormat( TLA700_MAIN_DATASET,
        components, TLA700_TEXT_SPACE, &bytesPerSample );
    if( !SUCCEEDED( hr ) )
    {
        DisplayError();
        return;
    }

    //
    // Get data.
    //
    VARIANT data;

    VariantInit( &data );

    hr = m_pLA->GetData( firstSample, numSamples, &data );
    if( !SUCCEEDED( hr ) )
    {
        DisplayError();
        return;
    }
    pDataArray = V_ARRAY( &data );

    // Get a pointer to the elements of the array.
    SafeArrayAccessData( pDataArray, (void HUGEP* FAR*)&pbstr );

    //
    // Add data to list control, with special icon for trigger sample.
    //
```

Data before and after Trigger:

```
  0:  0 E3 13 -500
  1:  1 E3 13 3500
  2:  1 E3 13 7500
  3:  1 E3 13 11500
T 4:  0 E3 12 15500
  5:  0 E3 12 19500
  6:  1 E3 12 23500
  7:  1 E3 12 27500
  8:  1 E3 13 31500
  9:  0 E3 11 35500
```

Run Control — Run Status: Idle

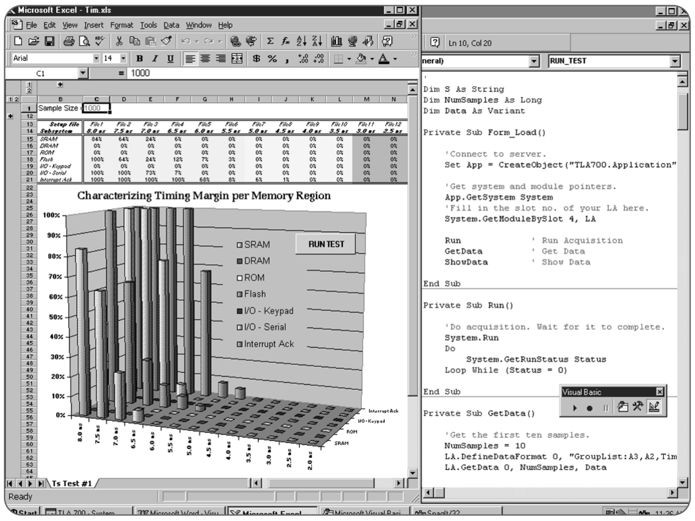
Load System — Path:

Status: Idle

C1: LA 2 — 9 — C2: LA 2 — 69 — Delta Time: 240ns

```
 Sample  CK0  A3  A2  Timestamp
      0   0   E3  13    10:14:44.999,999,999,500 s
      1   1   E3  13    10:14:45.000,000,003,500 s
      2   1   E3  13    10:14:45.000,000,007,500 s
      3   1   E3  13    10:14:45.000,000,011,500 s
      4   0   E3  12    10:14:45.000,000,015,500 s
      5   0   E3  12    10:14:45.000,000,019,500 s
      6   1   E3  12    10:14:45.000,000,023,500 s
      7   1   E3  12    10:14:45.000,000,027,500 s
      8   1   E3  13    10:14:45.000,000,031,500 s
      9   0   E3  11    10:14:45.000,000,035,500 s
```

▶ ***Figure 3.*** *Software front panel developed using Visual C++.*



Sample Size = 1000

| Setup file / Subsystem | File1 8.0 ns | File2 7.5 ns | File3 7.0 ns | File4 6.5 ns | File5 6.0 ns | File6 5.5 ns | File7 5.0 ns | File8 4.5 ns | File9 4.0 ns | File10 3.5 ns | File11 3.0 ns | File12 2.5 ns |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SRAM | 84% | 64% | 24% | 6% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| DRAM | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| ROM | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| Flash | 100% | 64% | 24% | 12% | 7% | 3% | 0% | 0% | 0% | 0% | 0% | 0% |
| I/O - Keypad | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| I/O - Serial | 100% | 100% | 73% | 7% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| Interrupt Ack | 100% | 100% | 100% | 100% | 68% | 8% | 6% | 1% | 0% | 0% | 0% | 0% |

```
'
Dim S As String
Dim NumSamples As Long
Dim Data As Variant

Private Sub Form_Load()

    'Connect to server.
    Set App = CreateObject("TLA700.Application"

    'Get system and module pointers.
    App.GetSystem System
    'Fill in the slot no. of your LA here.
    System.GetModuleBySlot 4, LA

    Run                 ' Run Acquisition
    GetData             ' Get Data
    ShowData            ' Show Data

End Sub

Private Sub Run()

    'Do acquisition. Wait for it to complete.
    System.Run
    Do
        System.GetRunStatus Status
    Loop While (Status = 0)

End Sub

Private Sub GetData()

    'Get the first ten samples.
    NumSamples = 10
    LA.DefineDataFormat 0, "GroupList:A3,A2,Tim
    LA.GetData 0, NumSamples, Data
```

▶ ***Figure 4.*** *Example plot of data acquired with the TLA700's TPI and exported to Microsoft Excel++.*

LAModule and DSOModule objects provide methods for module configurations, obtaining acquisition statistics and retrieving acquisition data. LAModule objects export a single interface called ILAModule; DSOModule objects export a single interface called IDSOModule. Refer to the sidebar TPI Methods for a complete list of the methods available from the TPI.

TPI documentation is available on the TLA700 as either on-line help or as a printable PDF file. Sample client programs are described in the documentation with source files shipped with the TLA700.

## Remote Control of a Logic Analyzer from a Computer

For remote control applications, developing a program with TPI is as straightforward as going through each test once while saving the logic analyzer configuration before each acquisition. Then with your favorite Windows program (e.g., Visual Basic, Visual C++, Excel, Delphi, LabView 5.0, etc.), develop your program to cycle through these same configurations, taking an acquisition and storing the data after each one. This simple, easy-to-use method can be used to quickly create even the most elaborate tests.

In the example shown in Figure 3, the user developed a simple software front panel using Visual C++ with the user's source code (shown in the background) to remotely control the TLA700 (standard user interface is shown below the user-developed software front panel). From the software front panel, the user can load a TLA700 system file and make an acquisition by pressing the "Run Acquisition" button. Note that the run status of the TLA700 is displayed on the software front panel. After the acquisition is complete, the acquisition data is retrieved and displayed in a grouped ASCII format. The trigger sample is marked in the display using a special icon.

As shown in Figure 3, the data in the software front panel is the same as the data shown on the TLA700's Listing display. A benefit of the TLA700's open Windows 95 platform is that the user's client program can be developed and tested directly on the TLA700. Once debugged and verified, the client program can remain on the TLA 700 or be moved to a remote PC connected via LAN which communicates with the TLA700 via DCOM.

The benefits of using TPI for this example include quickly developing a simple user interface with controls and data display area that could interact with the TLA700. All of this from within a familiar graphical programming environment such as that provided by Visual C++ that transparently communicates with the TLA700 via the COM interface – no complicated setup procedures or complex drivers to install. With the TPI, you can develop applications that range from simple and straightforward to sophisticated, yet powerful tests to remotely control your TLA 700 – all with tools that you're already familiar with.

## Using TPI for Remote Control with Support for Advanced Data Analysis

For advanced data analysis applications, the TPI is often used to extract data from the TLA700 logic analyzer to another application, either running on the TLA700 or on another computer connected to the network.

Support for advanced data analysis can best be illustrated with an example which will show how TPI can be used to perform advanced data analysis which isn't possible with any of the "canned" TLA700 data displays.

A hardware developer wants to validate the timing margins of all memory and I/O subsystems on the target system's processor bus. Will the target system always meet the required setup time for each subsystem? Will there be enough margin to handle temperature and component variations?

With the TLA700's TPI, you can create a program that automates this verification process of checking the setup time of every device on the processor bus. Once developed, you can stress test all of your proto-type target systems under a variety of test conditions before you sign the design off to manufacturing.

## TPI Methods

You can develop very powerful remote control client applications by going through the test on the TLA700 and saving the TLA700 setup before each acquisition. You can recall these setups from your client application. Then, using the TPI methods, you can customize these setups for your application. The methods for each of the four TLA700 objects include:

### IApplication

| | |
|---|---|
| `GetSystem()` | Get pointer to System object |
| `ShowWindow()` | Show or hide the TLA user interface |

### ISystem

| | |
|---|---|
| `GetNumModuleSlots()` | Number of slots in mainframe |
| `GetFirstModuleSlot()` | First slot number in mainframe |
| `GetSWVersion()` | SW version |
| `GetDiagCalStatus()` | Diagnostics and calibration results |
| `GetModuleTypeBySlot()` | Module type installed |
| `GetModulePropertiesBySlot()` | Module properties by slot |
| `GetModuleBySlot()` | Get module object |
| `GetModuleByName()` | Get module object |
| `LoadSystem()` | Load system from file |
| `SaveSystem()` | Save system to file |
| `Run()` | Start acquisition |
| `Stop()` | Stop acquisition |
| `GetRunStatus()` | Get acquisition status |

### ILAModule

| | |
|---|---|
| `LoadModule()` | Load module from file |
| `LoadTrigger()` | Load trigger from file |
| `SaveModule()` | Save module to file |
| `SetEventValue()` | Group, word, counter/timer value |
| `SetTriggerPosition()` | Set trigger position within acquisition memory |
| `GetNumSamples()` | Get number of acquired samples |
| `GetTriggerSample()` | Get sample number of trigger |
| `GetCounterValue()` | Get final counter value |
| `GetTimerValue()` | Get final timer value |
| `GetTriggerTime()` | Get time of trigger (referenced from acqui-sition start) |
| `GetBeginTime()` | Get time of first sample |
| `GetEndTime()` | Get time of last sample |
| `GetTimestampMultiplier()` | Get picoseconds per sample 'tick' |
| `DefineDataFormat()` | Define format of data to be returned |
| `GetBytesPerSample()` | Get bytes/sample in format defined |
| `GetData()` | Get acquisition data in format defined |
| `ExportData()` | Export acquisition data to file |

### IDSOModule

| | |
|---|---|
| `LoadModule()` | Load module from file |
| `SaveModule()` | Save module to file |
| `GetNumSamples()` | Get number of acquired samples |
| `GetTriggerSample()` | Get sample number of trigger |
| `GetTriggerTime()` | Get time of trigger (referenced from acquisition start) |
| `GetBeginTime()` | Get time of first sample |
| `GetEndTime()` | Get time of last sample |
| `GetDataRange()` | Get channel's vertical range (data) |
| `GetDataOffset()` | Get channel's vertical offset (data) |
| `GetDataSamplePeriod()` | Get sample period in picoseconds |
| `DefineDataFormat()` | Define format of data to be returned |
| `GetData()` | Get acquisition data in format defined |
| `ExportData()` | Export acquisition data to file |

For example, you could use Microsoft Excel to plot the resulting data and use Excel's Visual Basic for Applications to automate the process of characterizing the setup time margin on all devices on the processor bus (see Figure 4).

This Visual Basic application communicates with the TLA700 via the COM interface, loading specific setup files that program the TLA700 to count read cycles which violate a specified setup time. The application then sets the address range of each memory or I/O device. It also prompts the user to specify how many times to test for each read cycle. After each acquisition, it imports the counter values into Excel, then plots the data in a three-dimensional graph to show the distribution of timing margins for each memory region.

Using the TPI, you can further automate this process by running the same test on many different boards. You could then produce a graph or table that characterizes each board. You could also create various custom tools that allow the TLA700 to be used as a data acquisition front end. Using your own programs or popular commercial software, you can create advanced technical charts and graphs for analysis and presentation of results. The TLA700 can even be programmed to notify you by e-mail, paging, etc., after your analysis is completed.

## Conclusion

The TLA700's TPI represents the next-generation technology for remote control with support for advanced data analysis. Coupled with a modern logic analyzer architecture based upon an integrated PC running Windows 95, the TPI allows you to develop powerful data analysis applications. By tightly coupling TPI with a state-of-the-art logic analyzer instrument, your advanced data analysis applications can solve those elusive problems that threaten your product development schedules.

**Contact Tektronix:**

**ASEAN Countries & Pakistan** (65) 6356 3900

**Australia & New Zealand** (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** (Sony/Tektronix Corporation) 81 (3) 3448-3111

**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

For other areas contact Tektronix, Inc. at: 1 (503) 627-7111

**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**

Tektronix
Enabling Innovation