Interfacing Agilent Logic Analysers to MATLAB

Marcus Valentine

1 Introduction

This application notes serves as an introduction on how to interface the Agilent 168x/8xx/9x/9xx and 16700 series Logic Analyzers to MATLAB, using Microsoft COM (Component Object Model) automation. Familiarity is assumed with the Agilent's Logic Analyzer application; the concept of COM[1] and Mathwork's MATLAB.

The resulting MATLAB - Logic Analyzer system provides a powerful and versatile framework suitable for proof of concept and design verification of complex signal processing systems incorporating real digital hardware, in diverse applications such as wireless communications and medical.

2 Interfacing to 168x/9x/9xx series Logic Analyzers

The *Agilent Logic Analyzer* application includes the COM Automation Server which lets you write programs that control the *Agilent Logic Analyzer* application from remote computers on the Local Area Network (LAN). Agilent's *COM Automation Help* documentation contains programming examples for Visual Basic, Visual C++, Labview, Perl, Python and Tcl, but not, unfortunately, for MATLAB. The Visual Basic for Applications (VBA) examples and reference provide a good starting point to transform the VB code examples into MATLAB.

The Logic Analyzer application can be run on either the Logic Analyzer hardware, or the PC running the Matlab application[2]. It is recommended that the Logic Analyzer application be run on the PC running Matlab, as this removes the necessity of requiring a license for the Agilent *Advanced Customization Environment*[3] package. The technique described here does not require the *Agilent MATLAB Connectivity and Analysis Package* as this package requires the *Advanced Customization Environment*.

2.1 Connecting to the Logic Analyzer Application

In Visual Basic, we have:

```
Dim myConnect As AgtLA.Connect
Dim myInst As AgtLA.Instrument
Set myConnect = CreateObject("AgtLA.Connect")
Set myInst = myConnect.Instrument("localhost")
```

In MATLAB, connect to the local Logic Analyzer application thusly:

```
>> AGILENT_LA_SERVER_PROGID = 'AgtLAServer.Instrument.1';
>> logic_anal               = actxserver(AGILENT_LA_SERVER_PROGID);
```

This creates a COM server, and returns the COM object `logic_anal` representing the interface to the COM server. Inspect the properties of the object using the MATLAB ***get*** function:

```
>> get(logic_anal)
   Status: 'Stopped'
   Modules: [1x1 Interface.{91F3E657-040B-4499-9047-3461107FFD3F}.IModules]
   Tools: [1x1 Interface.{91F3E657-040B-4499-9047-3461107FFD3F}.ITools]
   Windows: [1x1 Interface.{91F3E657-040B-4499-9047-3461107FFD3F}.IWindows]
```

[1] See http://www.microsoft.com/com/
[2] Another possibility is to run Matlab on the Logic Analyzer hardware.
[3] Search agilent.com for B4606A.

```
   ...
```

Display the methods available on the interface using the MATLAB ***invoke*** function:

```
>> invoke(logic_anal)
   Close = void Close(handle, Variant(Optional))
   CopyDataToFile = void CopyDataToFile(handle, string, SafeArray(char),
Variant(Optional))
   DeleteFile = void DeleteFile(handle, string)
   ...
```

Retrieve the version of the Logic Analyzer application:

```
>> version_str = logic_anal.Version

version_str =

03.30.0002
```

2.2 Connecting the Logic Analyzer Application to the Logic Analyzer Hardware

We now have a connection between MATLAB and the Logic Analyzer application, but don't necessarily have a connection between the Logic Analyzer application and the Logic Analyzer hardware. First test for the connection using the ***IsOnline*** method:

```
>> [online_f status_str] = logic_anal.IsOnline

online_f =

     0
```

Make the connection using the ***GoOnline*** method:

```
>> LOGIC_ANAL_IP_ADDR = '192.168.1.99';
>> logic_anal.GoOnline(LOGIC_ANAL_IP_ADDR);
```

Test for online status once more:

```
>> [online_f status_str] = logic_anal.IsOnline

online_f =

     1

status_str =

192.168.1.99
```

Note that status_str has been set to the frame's name (as defined by the ***ComputerName*** property).

2.3 Creating additional handles

Create a handle to the modules object. This returns a collection of all the enabled hardware modules in the instrument.

```
>> modules = get(logic_anal, 'Modules')

modules =

      Interface.{91F3E657-040B-4499-9047-3461107FFD3F}.IModules
```

Retrieve the number of enabled hardware modules in the Logic Analyzer hardware by using the ***Count*** property:

```
>> num_of_modules = get(modules, 'Count')

num_of_modules =

     3
```

Use the ***Item*** property to get one of the objects in the collection either by index or name. This fails:

```
>> get(modules, 'Item', 1)
??? Invoke Error, Dispatch Exception:

Description: Invalid argument type.  Must be either an integer or string.
```

as the method is expecting the Item number to be of VB type long. A MATLAB double doesn’t work, but an int16 does. This is not obvious from the Agilent documentation, so some experimentation may be required.

This succeeds:

```
>> get(modules, 'Item', int16(1))

ans =

      Interface.{91F3E657-040B-4499-9047-3461107FFD3F}.IPattgenModule
```

2.4 Inspecting all the modules

This code fragment uses more properties to gather information on all of the enabled hardware modules:

```
for idx = 1:num_of_modules
   % Logic Analyzer modules start counting from zero, MATLAB counts from 1.
   module{idx} = get(modules, 'Item', int16(idx - 1));
   fprintf('%s\n', ['Slot ' , get(module{idx}, 'Slot'), ...
      ' Module: ',            get(module{idx}, 'Type'), ...
      ', model number ',      get(module{idx}, 'Model'), ...
      ', name ',              get(module{idx}, 'Name'), ...
      ', status ',            get(module{idx}, 'Status') ]);
end


Slot A Module: Analyzer, model number 16911A, name My 16911A-1, status Stopped
Slot B Module: Pattgen, model number 16720A, name My 16720A-1, status Stopped
Slot C Module: Pattgen, model number 16720A, name My 16720A-2, status Stopped
```

2.5 Retrieving data from the Logic Analyzer module

Create a handle to the Logic Analyzer module, this time by ***Name*** rather than by ***Index***, using the name discovered in the above code fragment.

```
>> My16911A  = get(modules, 'Item', 'My 16911A-1')


My16911A =

      Interface.{91F3E657-040B-4499-9047-3461107FFD3F}.IAnalyzerModule
```

Retrieving data from the Logic Analyzer module requires knowledge of the Bus/Signal set-up. Whilst the Bus/Signal set-up could be programmed over the interface, life can be made easier by instructing the hardware to load a previously saved Logic Analyzer configuration file, using the ***Open*** method. In this example, a preloaded configuration file has defined a Bus/Signal called ‘Ch A’.

Create a handle to the ‘Ch A’ Bus/Signal object:

```
>> ChA = get(My16911A.BusSignals, 'Item', 'Ch A')

ChA =

      Interface.{91F3E657-040B-4499-9047-3461107FFD3F}.IBusSignal
```

Run the Logic Analyzer using the ***Run*** method. Use of ***WaitComplete*** is recommended to avoid waiting until the end of time in the event of the trigger criteria not being met. In this example a timeout period of ten seconds is used.

```
logic_anal.Run
logic_anal.WaitComplete(int32(10))
```

See how much data we have:

```
>> CH_A_first_sample = ChA.BusSignalData.StartSample

CH_A_first_sample =

      -32768

>> CH_A_end_sample   = ChA.BusSignalData.EndSample

CH_A_end_sample =

       32767
```

Choose a DataType to return the data. In this example we will use AgtDataLong, which holds a maximum of 31 bits unsigned.

```
>> AGT_DATA_TYPE = 3;
```

Retrieve the data using the ***GetDataBySample*** method: In VB, an example call looks like:

```
Dim lArray() As Long
  lArray = _
     myData.GetDataBySample(myStartSample, myEndSample, AgtDataLong, myNumDataRows)
```

where the data is returned in lArray, and the number of results returned is in myNumDataRows.

In MATLAB, we need to move the assignment of the number of results to the left hand side:

```
[ChAData num_results] = ...
   ChA.BusSignalData.GetDataBySample(CH_A_first_sample, CH_A_end_sample, ...
      AGT_DATA_TYPE);
```

Generalising, in any VB code example where arguments are returned on the right hand side of the argument, these arguments need to be moved to the left hand side of the MATLAB assignment operator.

2.6 Interfacing to a Pattern Generator module

Interfacing to the pattern generator follows a similar strategy, creating a handle to the pattern generator, allowing access to objects further down the hierarchy. It is possible to use MATLAB to create Agilent PGB (Pattern Generator Binary) files on-the-fly. MATLAB can then instruct the pattern generator to load and play the PGB file.

2.7 Tidying up

When finished, it is good practice to clear handles from memory:

```
>> delete(logic_anal)
```

3 Interfacing to 16700 series Logic Analyzers

For the 16700 series logic analyzers, the COM automation server is included in the Agilent IntuiLink 16700 software. The PROGID for the Version 2 server is:

```
>> AGILENT_LA_SERVER_PROGID = 'Agt16700Ver2.Instrument';
```

Generally, interfacing follows the strategy outlined above; with the proviso many of the methods and properties have subtly different names. Refer to the documentation provided with the Intuilink software for details.

4 Coding Advice

When writing interfacing code, the following advice is offered:

- Investigate on the MATLAB command line, using MATLAB's ***get*** and ***invoke*** functions on new objects as they are created;
- Monitor what's happening by observing the Logic Analyzer application as commands are issued in MATLAB (a dual monitor set-up is recommended);
- Be wary of the types of both the arguments and returned data of function calls;
- Be wary of number of returned arguments – returned arguments may need moving to the left-hand side of the MATLAB assignment.
- Don't be afraid to experiment. You won't break anything[4]

5 Acknowledgements

The author wishes to thank Piers Glydon of Imperial Software, and Guy McBride of Agilent Technologies.

[4] The author absolves himself of all responsibility should any advice given in this document should, in fact, break something.