Introspect ESP

User Manual

Introspect ESP

User Manual

For Version 3.3

2013-05-04

Table of Contents

List of Figures

Introduction

Introspect's ultra-capable development environment allows you to easily and seamlessly develop and verify all your high-speed digital and mixed-signal algorithms. Designed for users with widely varying backgrounds and expertise, it offers an extremely intuitive interface simultaneously with extensible capability. Figure 1 shows a gallery of features that will be described in this document.

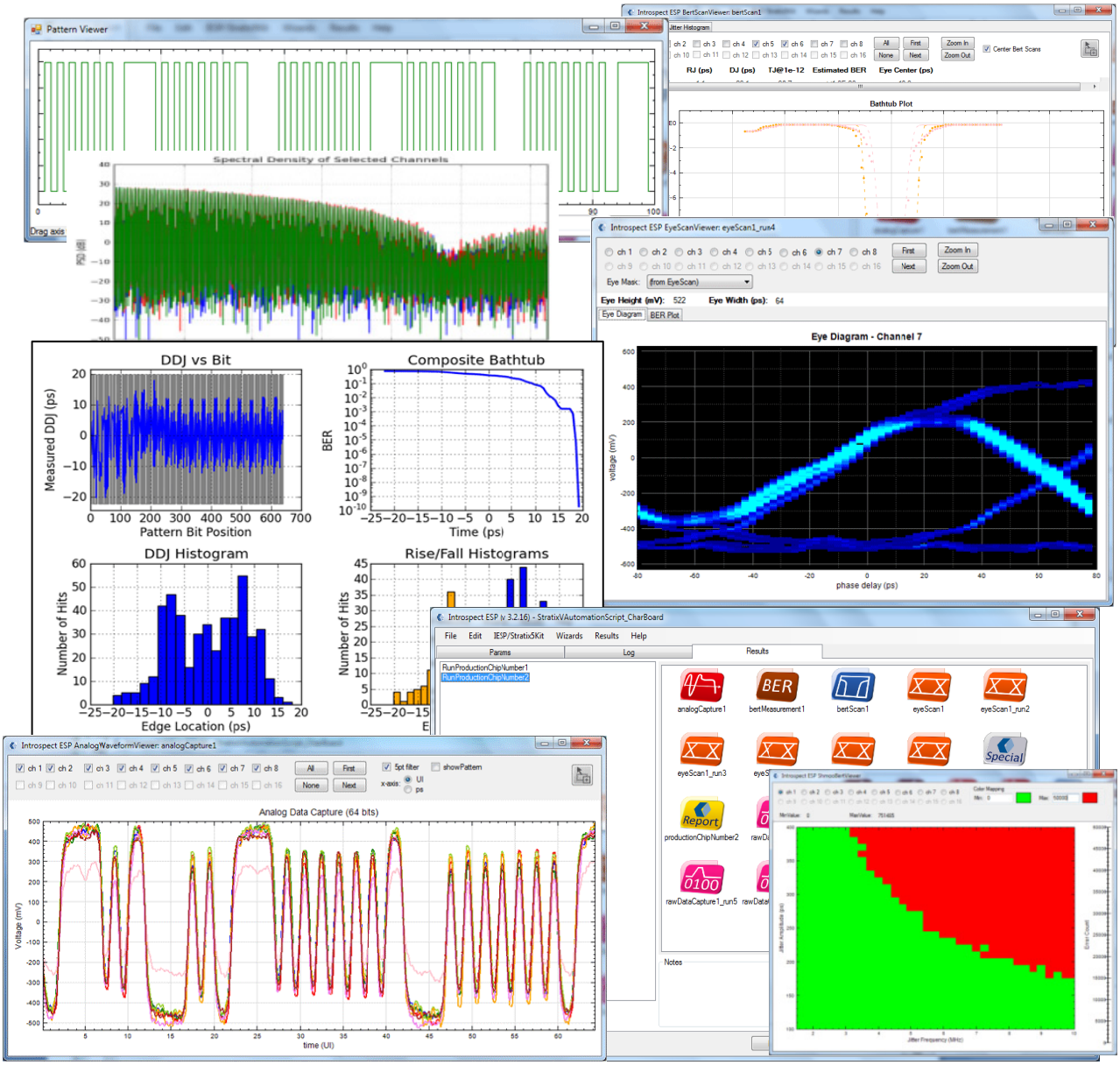

Figure 1 Mosaic of software features and results viewers.

Concepts & Terminology

The basic units that you will deal with when using the Introspect ESP software are

- IESP Test Processor
- Components
- Tests
- Results

IESP Test Processor

The "IESP Test Processor" (or just "IESP") is the combination of hardware and firmware that actually performs the measurements. The Introspect ESP software can be easily configured to use different hardware via the "formFactor" preference setting. Each IESP will have one or more "sub-parts", each of which will be connected to separately. The list of available sub-parts is specified via the "enabledSubParts" preference. (See the section on Customization/Preferences.)

Components

A "Component" is a conceptual encapsulation of some IESP functionality. It has parameters (properties) and actions (functions). A Component often contains or references other Components.

Some examples of Components:

- Pattern
- RxChannelList
- TxChannelList
- BertEngine
- BertScan
- Shmoo

The available channels will be formFactor-dependent. For example, the SV1C hardware has 8 RX and 8 TX channels. These are referred to in the software (in the RxChannelList and TxChannelList components) via channel numbers ranging from 1 to 8. For some purposes, the channels are grouped into "banks" which are referred to via bank numbers.

Tests

A “Test” is a conceptual encapsulation of the parameters and actions for a particular operation with the IESP. A Test usually makes use of one or more Components – these are considered as part of the Test entity. Usually the operation performed by a Test is a measurement of some sort.

You can create and edit Tests and save and load them from files. It is not necessary to be connected to the IESP hardware to create or edit Tests since that is done without any communication with the hardware.

If the computer is attached to the IESP hardware, a Test can be run, and the resulting measurement data are attached to the Test and are considered as part of the Test entity. Re-running a Test will add the latest measurement data to the Test (in addition to the data from previous runs).

There is a separate command-line utility “runSvtTest.py” that can be used to run previously saved Tests without the use of the GUI (e.g. for use with other 3rd-party applications).

The Test procedure uses the syntax of the Python programming language when invoking the component methods, but you don’t need to know anything about Python unless you want more fine-grained control.

Results

A “Result” is a conceptual encapsulation of data resulting from running a Test. As mentioned above, the Results from a Test are attached and considered as part of that Test. You can examine the data from a Result using the various data viewers provided with Introspect ESP. For example, the data from a BERT scan can be viewed as a bathtub plot.

Application Startup

When you launch the Introspect ESP application, the first window that appears looks like Figure 2.

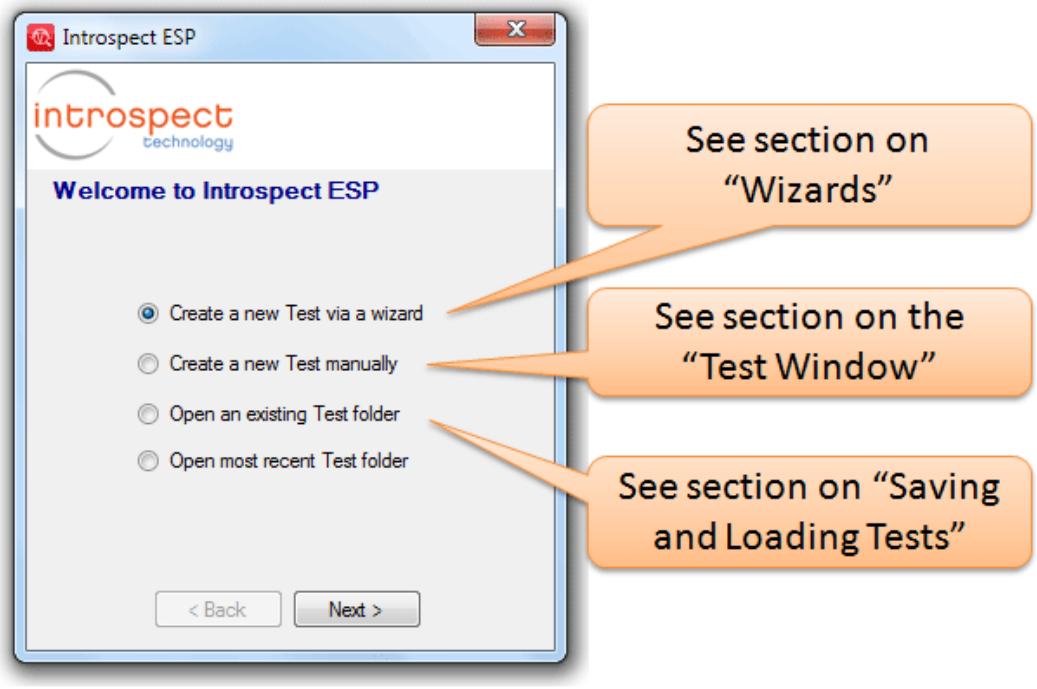

Figure 2 App startup screen.

This screen presents you with three options.

Choose the first option if what you are doing is one of the common operations supported by the available wizards (see the section on "Wizards").

If you are doing something more unusual, or more advanced, choose the option to create a Test manually (see the section on the "Test Window").

If you have a Test that you had saved previously that you want to open (to run it or to modify it), choose the third option (see the section on "Saving and Loading Tests").

Wizards

If you choose to run a wizard at application startup, you will see a window like in Figure 3.

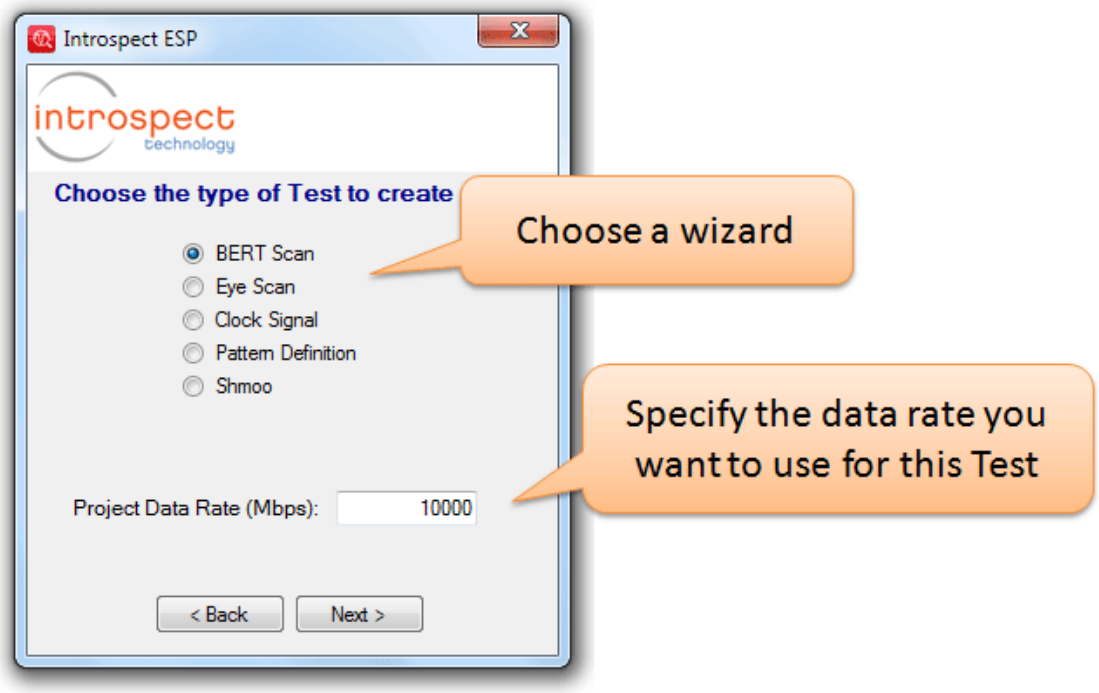

Figure 3 Wizard selection screen.

The following wizards are available at application startup (The same wizards are also available from the "Wizards" menu in any Test window.):

- BERT scan
  - Sets up measurement of bit error rate while scanning RX phase delay across a specified range
  - Allows you to specify expected bit patterns per channel
- Eye scan
  - Sets up measurement of bit error rate while scanning RX threshold voltage and phase delay across specified ranges
  - Allows you to specify expected bit patterns per channel
- Clock Signal

- Sets up clock signals with specified frequencies, duty-cycles, etc
- Pattern Definition
  - Allows you to define an arbitrary bit pattern for later use
- Shmoo
  - Allows you to define sophisticated multi-variable analysis sweeps in a matter of seconds

The "Project Data Rate" text field at the bottom of Figure 3 allows you to specify the data rate that you want to use for the Test. The data rate that you specify will be setup via the "globalClockConfig" component. (See the section on "Components" below.)

These wizards guide you step by step through the choices of key parameters for the task you have chosen. The wizards embed knowledge about the available ranges of these parameters and won't allow you to choose parameter values that would be inconsistent. As an example, Figure 4 shows the introductory stage of the BERT Scan wizard.

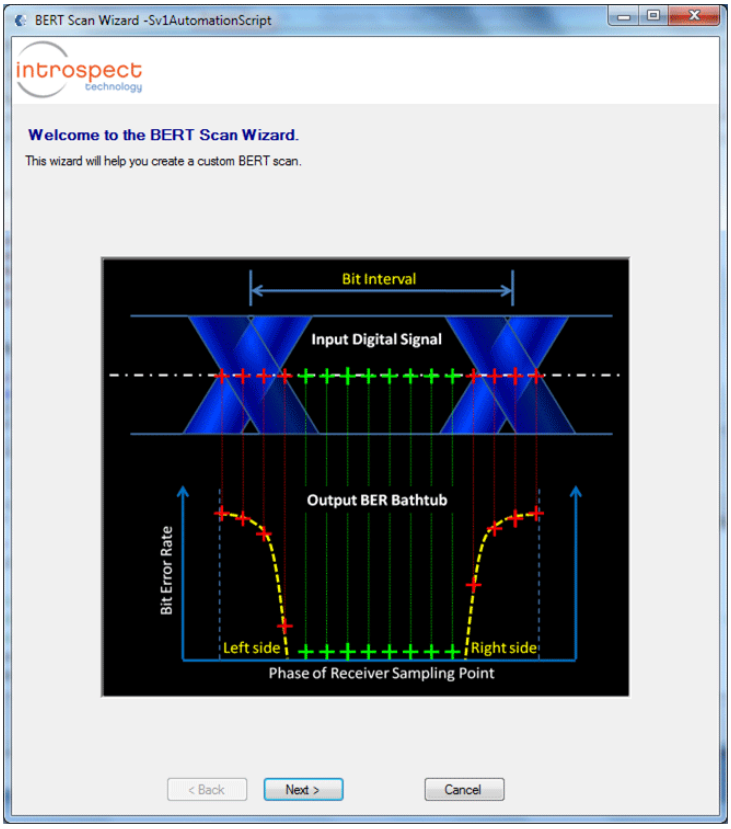

Figure 4 Introductory screen for the BERT Scan wizard.

The BERT Scan wizard knows what the possible range of RX phase delays is (for a specified data rate) and won't allow you to choose values outside of that range.

The end result of running a wizard is creation of one or more Components with parameters set for the specified task. If you started the wizard from the "Wizards" menu in the window for an existing Test, these components will have been added to that Test, otherwise a new Test will have been created with these components. You will see these components (each of which has a unique name) listed on the left side of the Test window. For example, Figure 5 shows what the components section of the Test window might look like after completion of the BERT scan wizard.

After the wizard has completed, you can run the Test (if your computer is connected to the IESP hardware) by pressing the "Run" button. See the section "Running a Test" below.

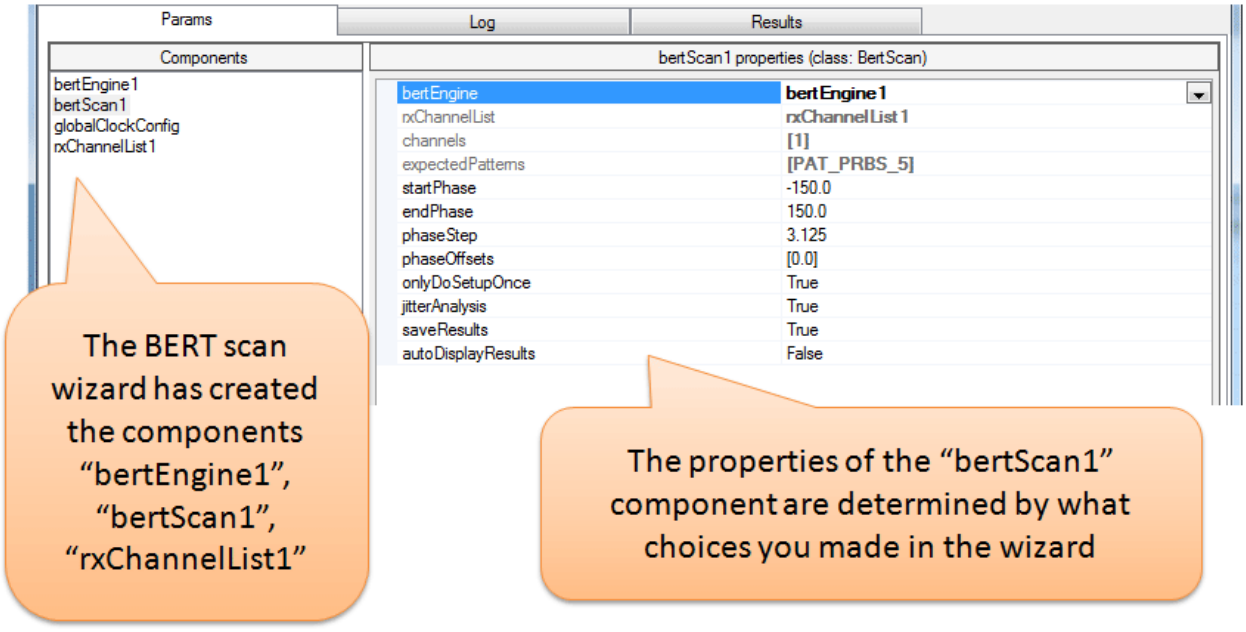

Figure 5 Generated components after stepping through the BERT Scan wizard.

Test Window

The “main” window of the Introspect ESP application is the Test window – a window that shows the components for a Test, and any Results from running that Test.

If you choose the option “Create a Test manually” from the initial window, you will get an “empty” Test window, which looks like Figure 6.

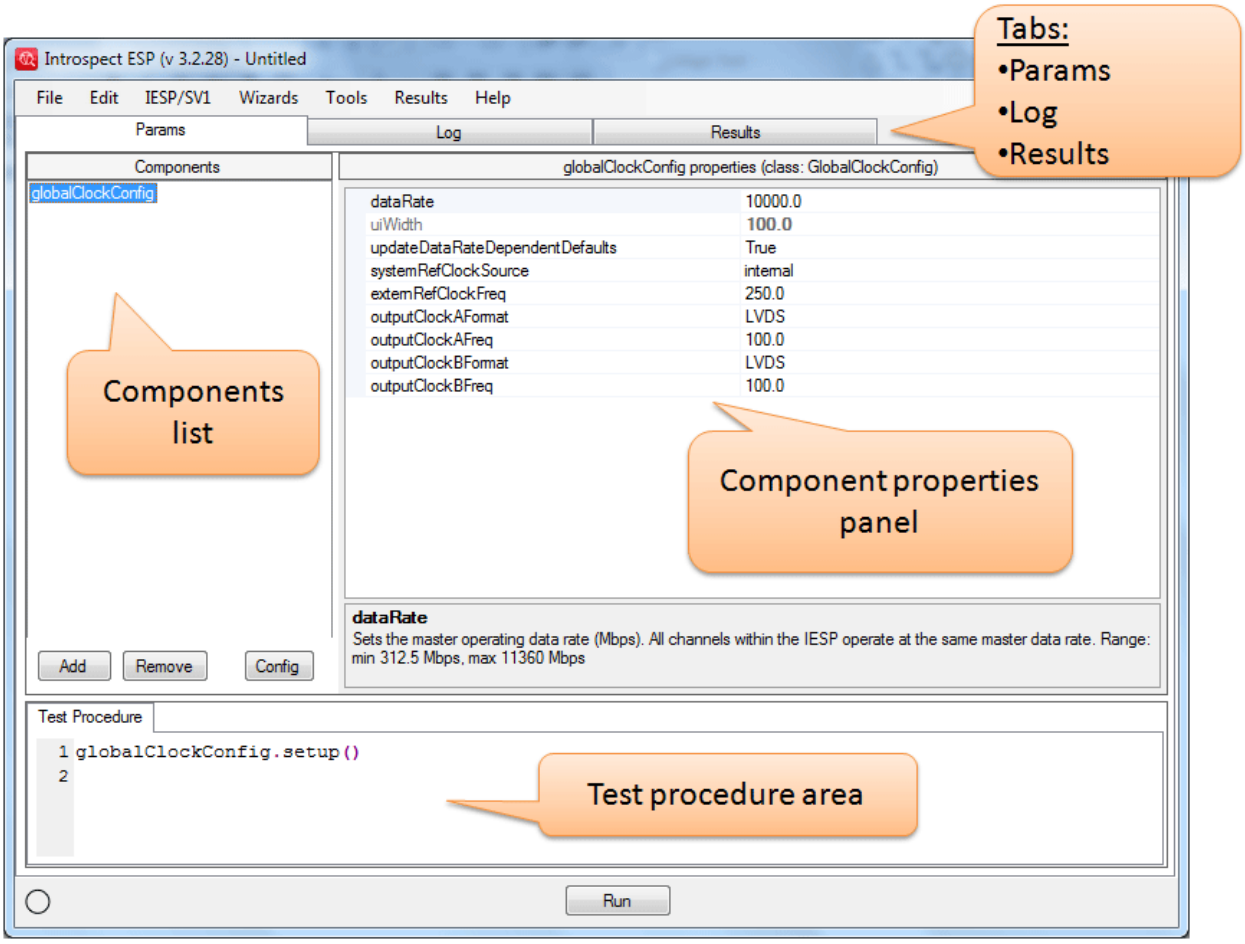

Figure 6 Screen capture of main test window.

There are three tabs in a Test window:

- Params
  - where you specify the parameters of the Test you want to run (via components and their properties)
- Log
  - where the log messages and error messages will appear when you run the Test
- Results
  - where the Results of the Test runs will be available for viewing

There are four regions of note in the "Params" tab of a Test window:

- Components List
  - shows a list of the names of the component instances in the Test
  - allows you to add, remove, or rename components
- Component Properties Panel
  - shows the properties of the currently selected component
  - allows you to edit those properties
- Test Procedure Area
  - shows the Python code that will execute when you run the Test
  - allows you to add extra Python code if desired
- Menus
  - provides access to other functionality, e.g. loading and saving of Tests

An "empty" Test has one component in it – the "globalClockConfig" which allows you to specify parameters like the data rate which apply throughout the Test. The "Test Procedure" area starts off with the command to "setup" the "globalClockConfig" component.

To configure the Test for your needs, you would:

- decide which components are required
- create those components (using the "Add" button below the list of components)
- set the properties of each component as desired (using the Properties panel)

When you use a wizard, the wizard creates the components and sets their properties according to the information you supplied.

Components List

Each component instance has a name (based on the name of its class) – e.g. the first RxChannelList created in a Test is called "rxChannelList1". The names are unique within the context of each Test. The name of the component instance becomes a Python variable that is used when referring to that component.

The names of all of the component instances in the Test appear in the Components list area on the upper left of the Test window. Figure 7 shows what that might look like in a more complicated Test:

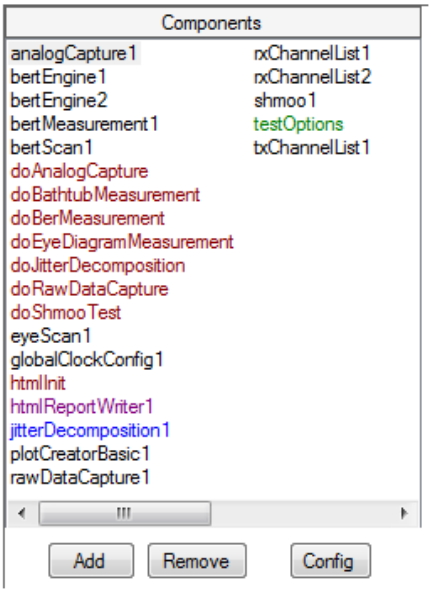

Figure 7 Components that are used in your Test.

When you click on the "Add" button that is below the list of components, the Add Component dialog appears as shown in Figure 8.

The available component classes are listed at the left of this dialog. When you click on one of the names on the left, the description of that component class appears on the right. (The "Components" section below gives more details on the available component classes.) If you click on the "Add Component" button in this dialog, an instance of the selected component class will be created and added to the Test.

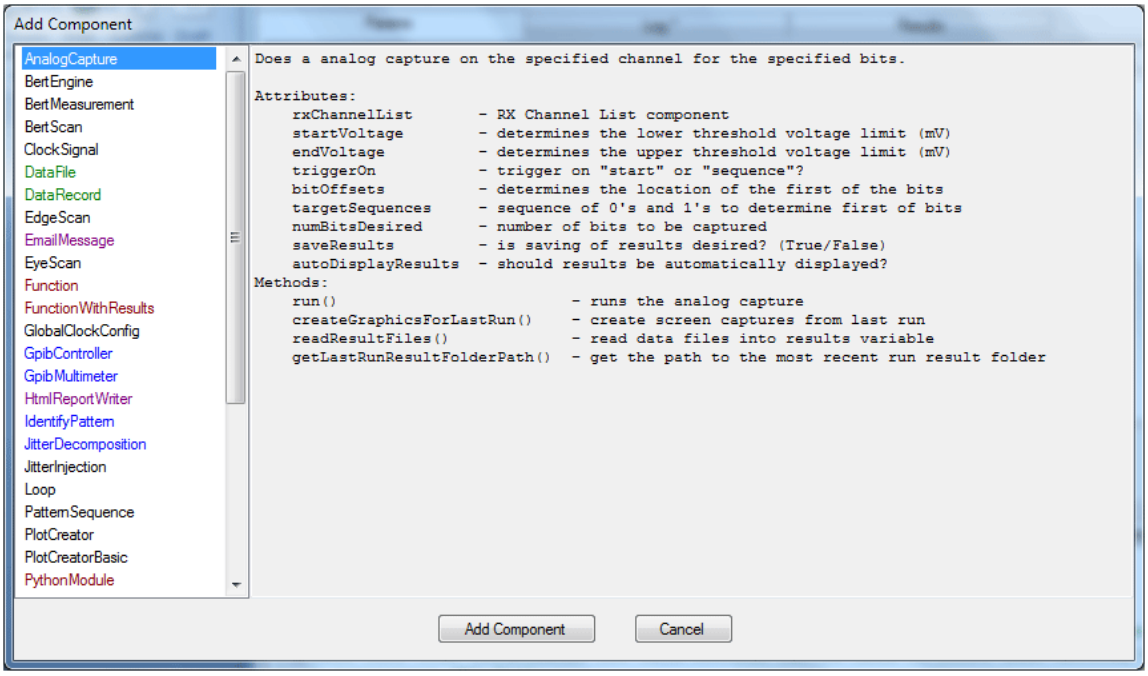

Figure 8 Add Component window.

If you decide that you no longer need a certain component instance, you can remove it by selecting its name in the Components List area and then clicking the “Remove” button. The component will be removed from the Test immediately – there is no confirmation dialog unless that component is referred to in the Test Procedure or by other components in the Test.

You can rename component instances by clicking on the name in the Component List area to select it, pausing slightly, then clicking again on the name to make it editable, and then typing in a new name.

You can duplicate a component by clicking on its name in the Component List area and holding down the mouse button and dragging it to a blank part of the Component List area. (The mouse cursor will change to show a little + sign while you are dragging it.)

If you have a second Test window open, you can drag a component from one Test into the Component List area of the second Test and a copy of the component will be added to the second Test.

Component Properties

To see the properties of a component instance, click on the name of that component in the Component List area on the left side of the Test window.

A newly added component will have default values for its properties. For example, the Properties panel for a newly created BertEngine component will look like Figure 9.

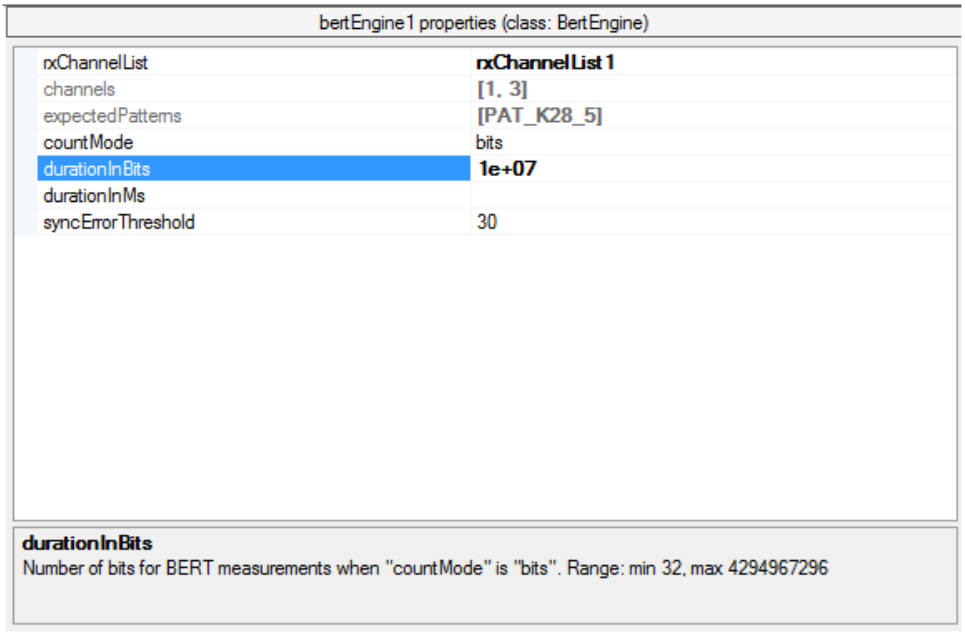

Figure 9 Basic BertEngine component properties.

The names of the properties are on the left; their values are on the right. The area at the bottom of the Properties panel shows a description of the property and usually indicates the units and what range of values is valid. Note that you can increase the height of this description area (via the thin bar just above this area) if it isn't large enough to display all of the text.

Some properties (e.g. the 'countMode' for a BertEngine) can only take one of a small set of values (enumerated types) – for these, a pull-down menu on the right side of the value provides a list of values to choose from. For these properties, you can cycle through the possible values by double-clicking on the value in the Properties panel.

Some properties (e.g. the 'channels' for an RxChannelList) are list-valued - i.e. you specify a list of values for these properties. A list in Python is indicated by a series of values separated by commas inside square brackets – for example: [1, 2, 3]. In the Introspect ESP Software, you can omit the square brackets when typing in these values and the brackets will be automatically added.

Some properties (e.g. the 'rxChannelList' for a BertEngine) are cross-references to other component instances – for these, the value is the name of the other component instance. A pull-down menu on the right side provides a list of the component instances that might be appropriate for such properties. Although not shown in Figure 9, the default for such properties is blank. You will need to fill this in with the name of an appropriate component instance before running the Test.

The Properties panel for a newly created RxChannelList component will look like the screenshot in Figure 10. The value for the 'channels' property is a list of channel numbers. The default value is [1] which is a list containing only one channel (channel #1).

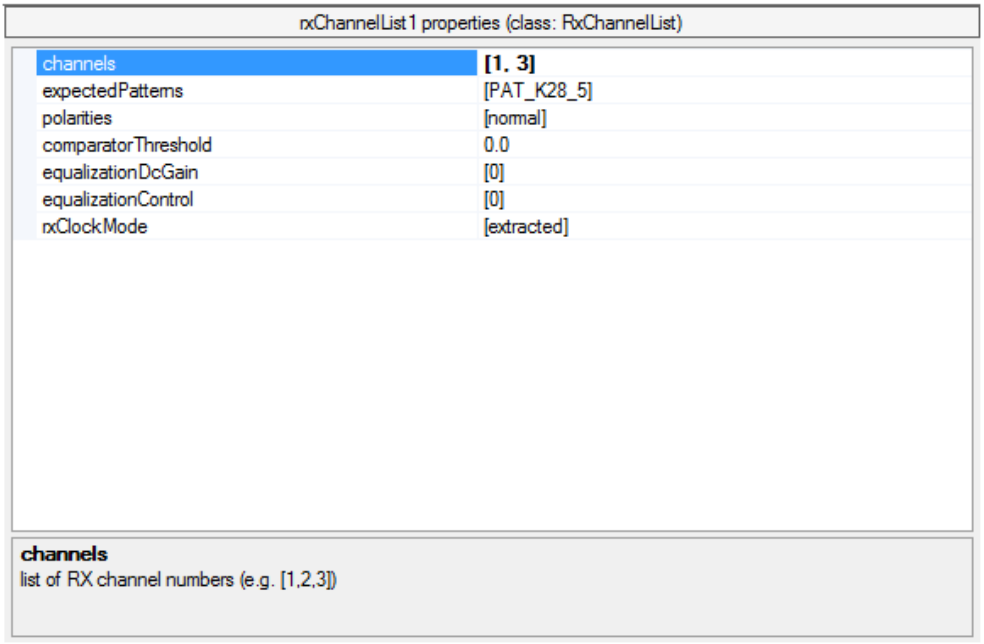

Figure 10 Properties for a RxChannelList component.

The value for the 'expectedPatterns' property is a list of pattern component names. The default value is [PAT_K28_5] which is a list containing only one pattern. PAT_K28_5 is one of the built-in pattern components (see the section "Components" below).

To make this RxChannelList component apply to channels 1 and 3, you would change the 'channels' property to [1, 3] as shown in the figure.

If the expected pattern for all these channels is PRBS31, you would change the 'expectedPatterns' property to [PAT_PRBS_31]. (If there are fewer patterns listed in the 'expectedPatterns' property than the number of channels, the last pattern applies to the remaining channels.)

Note that the ''expectedPatterns' property has a pull-down menu which allows you to append a pattern name to the end of the list.

After having made these changes, the Properties panel for the RxChannelList component will look like Figure 11. Note that the property values that are not the default values are shown in bold. If you want to set a property back to its default value, just erase the value entirely (i.e. enter a blank value) and it will be set back to the default value.

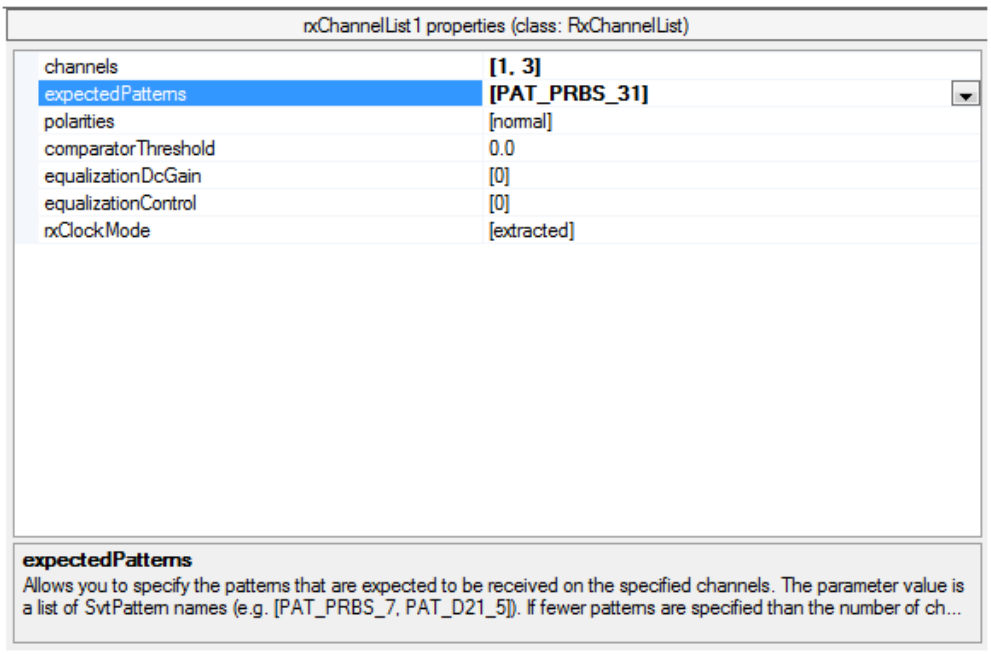

Figure 11 RxChannelList component after the expected pattern has been modified.

There is no need to press Enter after changing a property value – the change will take effect as soon as keyboard focus leaves that field. For example, you can change a property value and then immediately click the "Run" button.

Test Procedure Area

The Test Procedure area (at the bottom of the Test window) contains the Python code that executes when the Test is run (via the "Run" button).

Most components have either a "run" method or a "setup" method (see the "Components" section below) and a call to the appropriate method is entered automatically into the Test Procedure area (at the bottom of the Test window) when you add a new component instance to the Test. But the call to the "run" or "setup" method of a newly added component is entered at the *bottom* of the Test Procedure area and this may often not be what you want. For example, if you add a TxChannelList component, you might want that component to become active *before* you start a BertScan. If so, you will need to edit the lines in the Test Procedure area to get things in the right order.

For most Tests, reordering of the calls to the component methods is all that you will be likely to need to do. But you can insert arbitrary Python code in the Test Procedure area and this code will be executed when the Test is run. Read the "Python Commands and the Test Procedure" section below for full details, but a few basics about Python are useful to know:

Indentation is significant in Python and so it is best to avoid having extra spaces at the beginning of lines in the Test Procedure area.

You can comment-out a line by inserting a '#' character at the beginning of the line. For example, if you add a '#' at the beginning of the "bertScan1.run()" line so that it looks like: "#bertScan1.run()" (without the quotes), then the BertScan would not run.

The code in the Test Procedure area is "syntax highlighted" – i.e. it is coloured according to the Python syntactical constructs being used. In particular, comments are shown in green.

You can change the relative size of the Components List area and the Test Procedure area by resizing them by dragging the thin bars that are adjacent to these areas in the Test window. For example, you could make the Test window look like Figure 12 where more space is dedicated to viewing the Test Procedure.

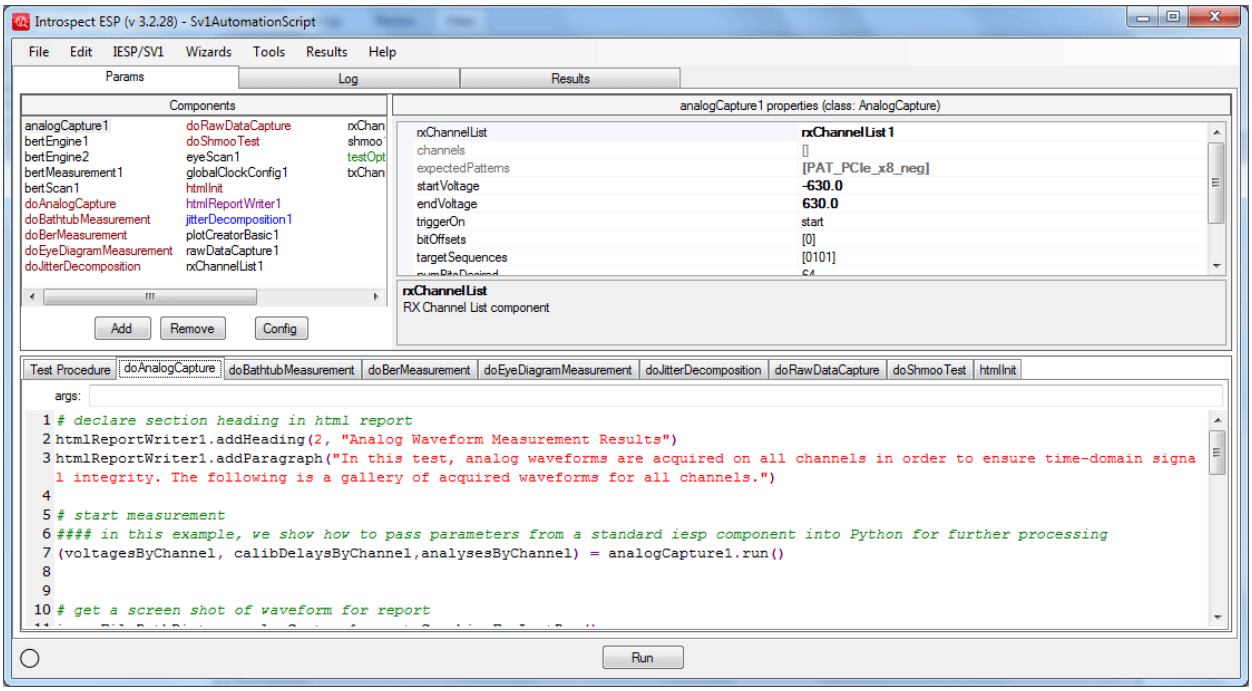

Figure 12 Test window showing a sophisticated Test and a large Test Procedure area.

Running a Test

You can run a Test by clicking on the “Run” button at the bottom of the Test window. In order to run a Test, you need to have your computer connected to the IESP hardware (via USB).

The first thing that happens when you run a Test is that the Introspect ESP program establishes a software connection to the command processors on the IESP hardware. This software connection is maintained after the Test has finished and is reused for subsequent Test runs. (If you want to disconnect for some reason, you can do so via the “IESP” menu.)

The Python code from the Test Procedure area is executed, which configures the IESP hardware with the values you specified in the components for this Test and runs the measurements specified. While the Test is running, the status indicator at the bottom left of the Test window flashes between solid green and hatched green. The “Run” button changes to a “Stop” button – clicking the “Stop” button will interrupt the Test run.

State	Status Indicator
Not Connected	
Connected and Idle	
Test running in this window	 ←flashing→ 
Test running in some other window	

Note that the status indicator is a software indicator and does not necessarily correspond to the activity lights on the hardware.

While the Test is running, messages from the components appear in the Log tab of the Test window. The Test window will switch automatically to the “Log” tab when you start a Test run so that you can see these messages. You can switch between the “Params”, “Log”, and “Results” tabs during a Test run. You won’t be able to change any of the component properties (of this or any other

Test), but you can look at previous results in the “Results” tab while waiting for the Test to finish.

After the Test has finished, the Test window will automatically switch to the “Results” tab so you can look at the results of the Test.

Results of a Test

The results from a Test are accessible in the “Results” tab of the Test window. Each time you run a Test, a new Result entry (named according to the current date and time) appears in the list on the left side of the “Results” tab. The Results are saved with your Test so you will have access to previous results when you open this Test some days later. Figure 13 shows what the “Results” tab looks like after several runs of a Test that has two calls to ‘bertScan1.run()’ in the Test Procedure.

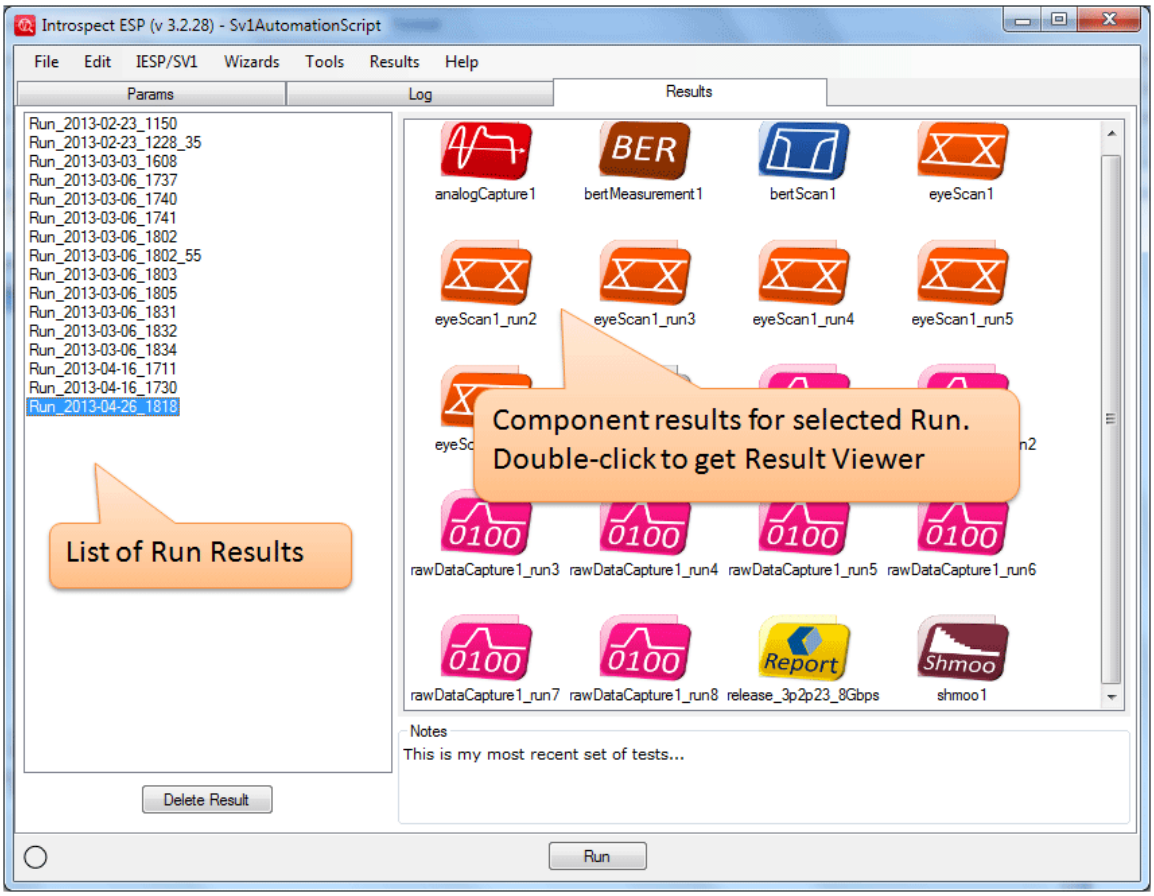

Figure 13 Illustration of the Results tab.

Selecting a Result in the list on the left shows the component results from that Test run in the area on the right. Usually there is one component result from each call to the "run" method of a component. The figure shows multiple component results from a Test run that was done at 18:18 on April 26, 2013.

You can rename a Result by clicking on the name in the list to select it, pausing slightly, then clicking again on the name to make it editable, and then typing in a new name. You may find it useful to use names that indicate something about the circumstances of the Test run.

You can delete Results that you don't need anymore by selecting them in the list and then clicking on the "Delete Result" button that is below the list. There is no confirmation dialog – the result data is deleted immediately.

If you double-click on one of the component results (e.g. "bertScan1" in Figure 13), the appropriate result viewer will pop up (as a separate window). You can keep as many result viewer windows open at a time as you like – this is often useful for comparing results. Figure 14 shows what the result viewer for a BertScan looks like.

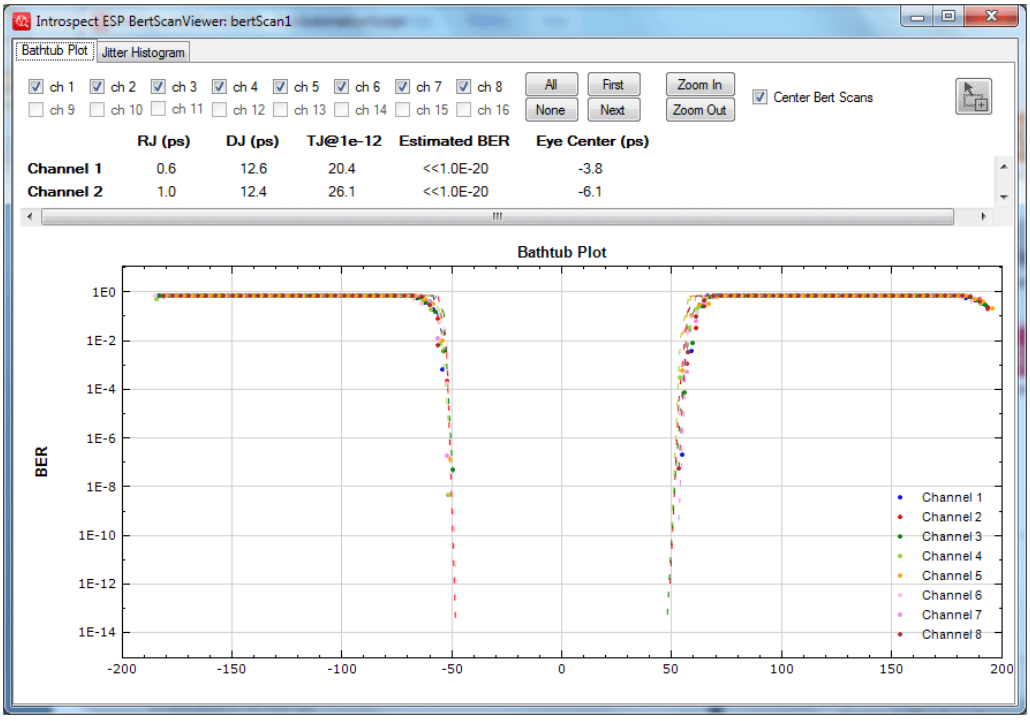

Figure 14 Bert Scan result viewer.

Each result viewer is different, but there are some commonalities:

- Select which channels you want to view using the checkboxes at the top. (The checkboxes for channels that weren't involved in this component are dimmed (unavailable).)
- Change the scale of the x or y axis (i.e. zoom in/out) by clicking the mouse in the axis area (where the numerical labels are shown) and then dragging (holding down the mouse button) along the axis.
- Move the graph around by clicking the mouse in the graph area and then dragging (holding down the mouse button).#

Accessing the raw data

The data from a Test run is saved on the filesystem (in a sub-folder named according to the current date and time) under the "Results" sub-folder of the Test folder. Most data is in CSV format and so you could examine this data in Excel if you wanted to do some analysis that isn't provided by the result viewers in Introspect ESP. Each component saves its data differently, but it is usual to have a separate file for each channel and to have the channel number as part of the filename.

An easy way to access the raw data is via the contextual (right-click) menus that are available over the Results in the list on the left side and over the component results on the right side. The "Show Folder" menu item opens the appropriate folder in Windows Explorer. The "Open CSV Files" menu item opens all the CSV files of a component result using whatever application is registered with Windows for the ".csv" suffix. These menu items are also available in the "Results" menu – they act on whichever Result is currently selected.

Python Commands and the Test Procedure

The “Test Procedure” is the Python code that executes when the Test is run. Most components have either a “run” method or a “setup” method (see the “Components” section below) and a call to the appropriate method is entered automatically into the Test Procedure when you add a new component instance to the Test.

You can add in calls to other component methods, and comment-out (or remove entirely) any code that you don’t want to run. Or add in any arbitrary Python code.

The most common reason to edit the Test Procedure is just to reorder the calls to the component methods so that things happen in the right order. For example, if you are using some of the IESP’s TX channels to output patterns to train your DUT, you need the ‘setup’ call for the TxChannelList to come before the call to the ‘run’ method of the EyeScan.

The rest of this section gives some examples of more advanced things you can do – it can be skipped over on a first reading of this manual.

Commenting out sections of code

To add a comment (for yourself or others) in the Test Procedure, you can use the Python comment character ‘#’ – this makes Python ignore anything after that character on that line. This is often used to “comment-out” lines of code that you don’t want executed (but want to keep for possible future use).

Another way to comment-out lines of code (or just to supply multi-line comments for yourself) is to use Python’s triple quoting mechanism. If you have 3 quote marks next to each other, Python will ignore everything after that until the next occurrence of 3 quote marks in a row. You can use either the single quote character (‘) or the double quote character (“) but you can’t mix and match. Figure 15 shows an example using double quote characters.

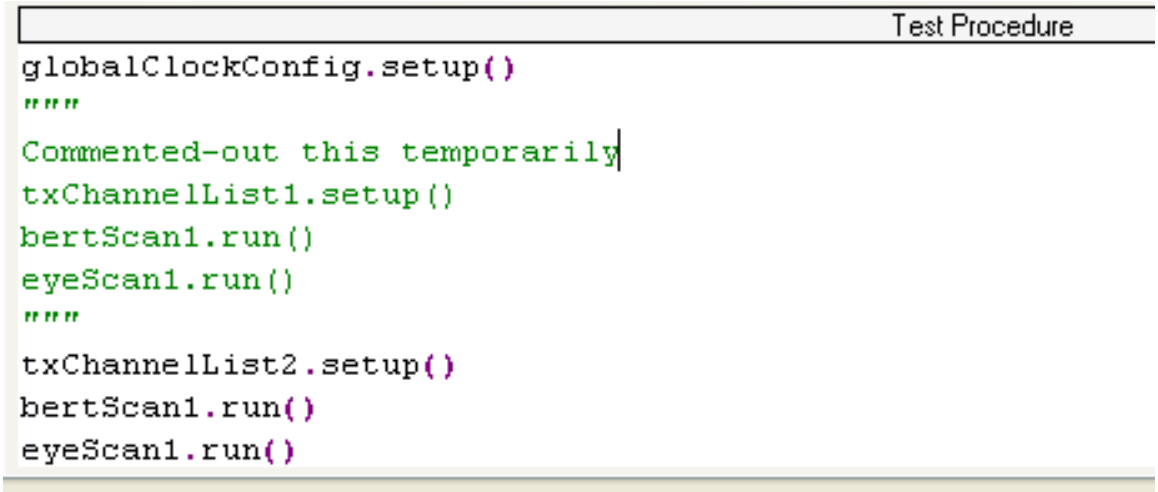

Figure 15 Illustration of the mechanism for commenting out sections of code.

Printing messages

You can use the Python 'print' command to output messages to the log. For example:

```
print "Starting second section of the test"
```

To print the value of some variables along with your message, use "%s", "%d", or "%g" as placeholders (for variables of type string, integer, or float respectively) and then supply the variables in a parenthesized list after a %. For example:

```
print "Iteration %d Value: %g" % (i, val)
```

Assigning values to variables

You can assign values to variables of your choosing and then use those variables later in the Test Procedure. Many of the component methods return a value and it is often useful to assign this return value to a variable. For example, the 'run' method of BertScan returns a tuple (calibDelays, errCounts, jitterAnalysisResults) – this is not usually used since these results can be viewed via the BertScan viewer, however you might want to do some specialized analysis on the data – something like this:

```
(calibDelays,
 errCounts,
 jitterAnalysisResults) = bertScan1.run()
analyzeBertData(calibDelays, errCounts)
```

Importing other Python code

The implementation of the function “analyzeBertData” in the above example might be in a file of Python code that you wrote earlier. Instead of copy/pasting this code into the Test Procedure, you can make it available by using the Python ‘import’ command – for example:

```
from myPythonStuff import *
```

In order for this to work, the file “myPythonStuff.py” must be somewhere in the Python search path. Note that within Introspect ESP, the parent folder of the “dftm” folder is in the search path, so if you put your Python code files into that folder, they will be found.

Changing properties of components

You can change properties of a component programmatically by assigning to the properties and then calling either the ‘setup’ or ‘update’ method on the component. (The ‘setup’ method sends all of the component property values to the IESP hardware, while the ‘update’ method only sends the values that have changed.)

For example:

```
txChannelList1.voltageSwings = [800, 600]
txChannelList1.preTaps = [0]
txChannelList1.update()
```

Looping

You can loop (iterate) over sections of code by using the Python looping constructs ‘for’ or ‘while’. (But note also that there is a “Loop” component available to make this easier.) For example:

```
for i in range(0, 3):  # i will be 0, 1, 2
    print "%d) Hello" % (i)


myPatterns = [PAT_DIV20, PAT_K28_5, userPattern1]
for i in range(0, 3):
    pattern = myPatterns[i]
    print "Using pattern %s" % (pattern.name)
    rxChannelList1.expectedPatterns = [pattern]
    rxChannelList1.update()
    bertScan1.run()
```

Defining functions

You can define functions to encapsulate sections of code. For example, as an alternative to the first loop above, you could define a function:

```
def printHello(n):
    for i in range(0, n):
        print "%d) Hello" % (i)
```

and then call it like:

```
printHello(3)
```

A more interesting example (a simple version of what is done in the IdentifyPattern component):

```
def tryPattern(pattern):
    print "Trying pattern %s" % pattern.name
    rxChannelList1.expectedPatterns = [pattern]
    bertEngine1.setup()
    synchedChannels = bertEngine1.syncWithCdr()
    if synchedChannels:
        return True
    else:
        return False

for pattern in [PCIe1, PCIe2, PCIe4, PCIe8]:
    if tryPattern(pattern):
        print "detected pattern %s" %
pattern.name
        break
```

Note that the component variables (e.g. 'bertEngine1') are effectively global variables in the Test Procedure and hence are available inside functions that you define.

Inserting delays

You can pause the execution of the Test Procedure for a specified number of milliseconds by using the 'sleepMillis' function – for example:

```
sleepMillis(1000) # one second delay
```

Waiting until ready

You can pause the execution of the Test Procedure until something else is ready via the 'waitForOkDialog' function – for example:

```
waitForOkDialog("Click OK when DUT is ready")
```

Running a shell script

You can run a shell script or other Windows executables via the 'runShellScript' function. This could be used to control other test equipment, or to do data analysis. For example:

```
runShellScript("perl myScript.pl")
```

Sending email or texting

You can send an email message at any point in the Test Procedure by using the 'send' method of the EmailMessage component. For example:

```
emailMessage1.send("Partial Result",
                   "value: %.3f" % value)
```

Note that most cellular phone companies provide a way to send text messages (SMS) to a phone via email (see for example: http://www.makeuseof.com/tag/email-to-sms/) and thus the EmailMessage component can be used to send text messages to your phone.

PyLab: SciPy and MatPlotLib

The advanced functionality of SciPy and MatPlotLib is available for use in the Python code you write in the Test Procedure or in Function components. All of the names from the 'pylab' module are automatically available in the environment used for running Tests. This means that the environment is similar to that obtained with the "--pylab" option to iPython.

Lower-level access to the IESP hardware

Usually you control the IESP hardware via the facilities provided by the components. But lower-level access is available via the 'iesp' object that you can obtain by calling IESP.getInstance() - for example:

```
iesp = IESP.getInstance()

iesp.setCdrModeEnabled([1, 2, 3], True)
```

For more info on the functions mentioned see the documentation in the files "svt.html" and "iesp.html" (in the folder "Doc"). To learn more about Python, see the documentation at http://www.python.org

Saving & Loading Tests

The Introspect ESP application is a document-based application like Microsoft Word. Each Test window is a separate document and can be saved and loaded independently. You can have several Test windows open at the same time.

When you create a new Test, the associated parameters and data are either kept in RAM or in a temporary folder on your hard disk. This allows you to do quick "one-off" experiments without being bothered about filenames, etc. But usually you will want to save your Tests (and associated Result data) for later use. You do this via the "Save" menu item in the "File" menu. A saved Test is a folder (containing sub-folders and files) so the "Save" menu item will prompt you for the name and location of a folder to save the Test in. After you save the Test, the name of the Test folder will appear at the top of the Test window.

If you want to keep the current version of your Test while continuing to modify the parameters, use the "Save As..." menu item in the "File" menu. This will create a copy of the Test in a different folder.

To load a previously saved Test, use the "Open..." menu item in the "File" menu. You will be prompted to choose the Test folder to open.

Structure of a Test folder

A Test folder always has sub-folders "Params" and "Results". If you have enabled writing of logs to a file (see the "Customization/Preferences" section below), then there will also be a sub-folder "Logs".

The "Params" folder usually only has one file: "testProcedure.py". This file contains Python code to create the components of the Test, followed by the code of the Test Procedure. If you are careful to get the syntax correct, you could edit the "textProcedure.py" file in a text editor (e.g. "Notepad++" or "Komodo") – this is especially useful if you have several Tests that you want to change in the same way.

The "Results" folder is initially empty but each time you run the Test, a new sub-folder is created there (by default these folders are named according to the date/time) to hold the results of the run.

Each run results sub-folder contains:

- A snapshot of the “testProcedure.py” file – this file contains the state of the Test parameters used to generate the associated results data. It might be useful if you have changed the Test parameters after that run and want to revert to what it was at that time. (You could revert by manually copying the “testProcedure.py” file from the run results folder to the “Params” folder.)
- One or more component results sub-folders, each of which contains:
- A file “.resultInfo.csv” with info about the component results. This file is normally hidden since its name starts with a dot.
- One or more CSV files with the raw data from the component’s ‘run’ method. Usually there is one file per channel and the channel number is part of the filename.
- The “Logs” folder (if enabled) contains the log files with the messages that appear in the Log tab. There is one log file for each session, named with the date/time of the start of the session.

If you are short of disk space, you can remove old log files and any run results that you no longer need.

Components

The Introspect ESP software loads components (implemented in Python) from the three sub-folders of the “dftm/components” folder. The “basic” sub-folder holds the general-purpose components – these components are often used in the implementation of other components. The “extra” sub-folder holds special-purpose components like those for DisplayPort testing. The “user” folder holds user-defined components. (Documentation on how to create user-defined components is available upon request.)

Each component has a number of properties (attributes in Python) and a number of methods. The component properties are shown in the Properties panel of the Test window.

Most basic components have a ‘setup’ method and an ‘update’ method. The ‘setup’ method sends all of the component property values to the IESP hardware. The ‘update’ method sends only those property values that have changed since the last ‘setup’ or ‘update’.

Components that produce a result (measurement data) have a ‘run’ method. The ‘run’ method often calls the ‘setup’ method internally. The ‘run’ method usually returns a value but the return value is not usually used in the Test Procedure since the data has also been written to files which are used by the various Result viewers.

Full documentation on the component properties and methods is in the file “svt.html” (in the folder “Doc”). Note that the components’ Python class names usually start with “Svt” but this prefix is omitted in the component class names listed in the GUI.

The two most basic component classes are RxChannelList and TxChannelList. An RxChannelList instance represents a selection of the RX channels of the IESP. Components that do measurements on incoming signals usually refer to an RxChannelList component. Such components usually invoke the ‘setup’ method of the RxChannelList component within their ‘setup’ method and thus it is not usually necessary to explicitly call the ‘setup’ method of an RxChannelList component in the Test Procedure.

A TxChannelList instance represents a selection of the TX channels of the IESP. TxChannelList components are usually used to generate outgoing signals to be sent to the DUT. Since they are not (usually) referred to by some other component, it is usual to

have a call to the ‘setup’ method of the TxChannelList component in the Test Procedure.

There is always a component instance named ‘globalClockConfig’. This component sets up the IESP data rate and other clock parameters via the call to its ‘setup’ method which is at the start of the Test Procedure.

If you use one of the wizards, a series of components will be created and linked together automatically. But it is easy to get the same thing by creating the components via the “Add Component” dialog and linking them up manually using the menus in the Properties panel. For example, to get the same thing as what the “Eye Scan” wizard produces, add an RxChannelList component, a BertEngine component, and an EyeScan component, and then link the BertEngine to the RxChannelList component and link the EyeScan to the BertEngine component. It doesn’t matter which link you do first. All that matters is that the cross-references exist when you run your Test – if they don’t, you will get a runtime error message.

It is quite common (and recommended) to have several components sharing a reference to a component. For example, the references might go like this:

bertScan1 -> bertEngine1 -> rxChannelList1

eyeScan1 -> bertEngine1 -> rxChannelList1

analogCapture1 -> rxChannelList1

Running Tests from the Command Line

It is sometimes useful to run Tests from the command line instead of from within the Introspect ESP GUI. For example, you might want to run a Test from LabVIEW, or in some other context. Since the parameters of a Test are saved as Python code in the "testProcedure.py" file (in the "Params" sub-folder of the Test folder), this is easy to do.

The Python script "runSvtTest.py" supplied in the "Python" folder can be used to run a saved Test from the command line. If you look at that script, you will see that it does three things:

Loads the Test from the given folder using:

```
test = SvtTest.load(testFolderPath)
```

Connects to the IESP hardware using:

```
iesp.connectViaFtdi()
```

Runs the Test using:

```
test.run()
```

Customization/Preferences

When the Introspect ESP GUI starts up, it reads the file "IntrospectESP_GUI.ini". This file contains preference settings. Here are some of the more commonly changed preferences:

- formFactor
    - This specifies which IESP hardware/firmware is to be used. For example, to use the SV1C hardware, you would set this preference to "SV1" (without the quotes).
- enabledSubParts
    - This specifies which sub-parts of the IESP hardware are available. For example, with the DV1600 hardware, you would set this preference to "moduleA, moduleB" (without the quotes) if both modules are available. The Introspect ESP software will connect to each sub-part separately.
- testDefaultPath
    - This specifies the default save and load locations for all Tests. It follows the same format as specified above for 'pythonFolderPath'. This preference is disabled by default. To enable it, remove the ';' found at the beginning of the line. (The semi-colon is the comment character for INI files.)
- defaultDataRate
    - This specifies the default data rate (in Mpbs).
- writeMessagesToLogFile
    - If you set this to true (it defaults to false), the messages that appear in the Log tab will be written to a file under the Test folder.

There are other preference settings for customizing the colours of the Test window and for defining eye masks for use in the EyeScan viewer. Comments in the "IntrospectESP_GUI.ini" file explain how to use these.

The Test Procedure area has syntax highlighting supplied by the "ScintillaNet" .NET component (http://scintillanet.codeplex.com/). You can customize the colours

used by editing the file “ScintillaNet.xml” that is in the “GUI” folder.

The icons used on the right side of the “Results” tab are supplied as PNG files in the “ResultIcons” folder in the “GUI” folder. You could substitute different image files if you want to customize these icons. The size of the result icons is determined by the preferences “resultIconsWidth” and “resultIconsAspectRatio”.

Troubleshooting

Failure to connect to IESP

Check the messages in the Log tab. If the error message says something like “No FTDI devices found”, that likely means that your computer isn’t connected (by USB) to the IESP hardware. If the error message says something like “Command processor not responding”, that likely means that the IESP hardware isn’t powered on, or is hung and needs to be power-cycled.

BERT sync failure

First check that the ‘expectedPatterns’ in your RxChannelList component corresponds to the patterns of the incoming signals. Try reversing the ‘polarities’ in your RxChannelList component. Use the RawDataCapture component to grab a snapshot of the incoming signal and check that it is as expected. Use the AnalogCapture component to check signal integrity. If the input signal is noisy, try increasing the ‘syncErrorThreshold’ value in your BertEngine component.

Contacting customer support

Please send a detailed description of the problem (including a copy of the Test folder if possible) to customer support:

support@introspect.ca

Introspect Technology
195 Labrosse Avenue, Pointe-Claire
Quebec, Canada H9R 1A3
http://introspect.ca