Errata

Title & Document Type: **E4859A Serial Cell Generator and Analyzer System Quick Start Guide**

Manual Part Number: **E4859-99003**

Revision Date: **January 1, 1989**

About this Manual

We've added this manual to the Agilent website in an effort to help you support your product. This manual provides the best information we could find. It may be incomplete or contain dated information, and the scan quality may not be ideal. If we find a better copy in the future, we will add it to the Agilent website.

HP References in this Manual

This manual may contain references to HP or Hewlett-Packard. Please note that Hewlett-Packard's former test and measurement, life sciences, and chemical analysis businesses are now part of Agilent Technologies. The HP XXXX referred to in this document is now the Agilent XXXX. For example, model number HP8648A is now model number Agilent 8648A. We have made no changes to this manual copy.

Support for Your Product

Agilent no longer sells or supports this product. You will find any other available product information on the Agilent Test & Measurement website:

www.agilent.com

Search for the model number of this product, and the resulting product page will guide you to any available information. Our service centers may be able to perform calibration if no repair parts are needed, but no other support from Agilent is available.

Agilent Technologies

Getting Started Guide

HP E4859A Serial Cell Generator and Analyzer System

HEWLETT® PACKARD

About this Guide

This guide gives application independent examples of how to use and verify the HP E4859A system.

This guide is designed so that you can go through the examples step by step. Each task is explained independently of the others allowing you to use this guide also as a quick reference.

Verify Installation

Using a simple adder (or your own DUT), an oscilloscope (optional), some coaxial cables and a BNC T-piece, verify the operation of the system.

Test Examples

Three examples take you through all the basics of using the HP E4859A.

Revision History

Document Edition	For Software Release	Comments
1.0 (E1195)	1.1x	First document and product release
1.2 (E0196)	1.2x	New Timing and Main window, Frame and PRBS infinite length (continuous PRBS), cells from AUX and MAIN outputs as controls linked to generators or start of frame, link timing in bits and seconds, etc
1.3 (E0496)	A 1.3.x	Multiple cells from one generator, a second method for entering guard-time, and a C/C++/VEE programming interface.
2.4 (E0696)	A 2.4.x	Asynchronous analyzer modes.

Contents

Contents

Contents

Contents

Chapter 1

Verify Installation

Verify Installation

There are several levels of installation verification:

1 Operating platform up and running

After switching the system on, the VXI controller module (e.g. V743) successfully boots from its hard disk and the operating system software is running. Check no failure lights are illuminated on the controller and the HP VUE Front Panel is displayed on the monitor.

2 Inter-module VXI communication

Check the front panels of the modules before starting the user interface. If failure to communicate with the VXI bus has occurred, then the Failed indicator will be illuminated.

3 Module hardware self tests

Check the front panels of the modules before the user interface is started. Hardware failure is reported by REGULAR (not random) flashing of the Access indicator. If the hardware test has failed see Troubleshooting in the Installation Guide.

4 System Selftests

After the software is started, the user interface allows some tests to:

- check the proper function of the modules (each module performs its own built in self test).
- check module plugging (are the modules plugged side by side, is the VXI local bus configured right?)
- check front panel wiring (are all modules connected to the central module, are there signals swapped / interrupted / shorted?)

The following tests can be run from the Admin->Self Test menu option in the Main Connection window:

a Read Module Error Queues:

This test helps determine the problem if the system is behaving strangely. You can find if a module has a problem or if it received a wrong command. Under normal operating conditions the error queues are empty.

b Module Selftest:

All modules perform their built in self tests independently. This may take up to 1 minute. During this time the modules do not accept commands. An attempt to program new values during this time may cause a time-out error.

c System Selftest:

The system configuration is checked. First, that there are no gaps between the system modules, which would mean a local bus interrupt (check module position in the frame). There is also a check if the left most module is a central clock module.

The local bus is checked. The sequencer clock signal is set high and then low and the result is checked on every output module.

The front panel wiring of the system clock cables is checked. The most likely error is a forgotten cable. Missing, broken and cross wired cables are detected and the result is displayed.

d 10 MHz Performance Test:

The frequency accuracy is a hard specification. For an easy setup in the field, this menu item sets the trigger output to a frequency of 10 MHz and to an amplitude of 1V AC, which should be suitable for the most frequency counters.

Note: Be sure that the measuring equipment used is accurate enough for the measurement. There is no sense measuring a +/- 50 ppm performance with an RC time base oscilloscope!

The self tests generate an output on a display window, containing all the important information about frame, slot number, module number including options and related error outputs. A successful check is reported with the message “No Error” to indicate that the test has been performed for this module and has not been suppressed or forgotten.

All results can be printed (if a system printer is available) or be stored to disk. This eases diagnostics in case of malfunction.

Such files can then be mailed for support.

5 Application test

Using the test setup and an example described in this guide verify your system by loading the setup from the database and running it. You need only perform this test if you suspect the system is not functioning correctly

Table 1 **Database Setup Names for Getting Started Examples**

Getting Started Example	Database Setup Name
Example 1	gsm_example_1
Example 2	gsm_example_2
Example 3	gsm_example_3

To perform an application test:

1 Choose the example that you want to perform

2 Connect hardware as described in the Test Setup

3 Use the Open Setting option in the Main window File menu to recall the corresponding database setup.

4 Run the test as described in the chosen example

5 Verify the test as described in the chosen example

Chapter 2

Test Examples

About these Examples

In order to become familiar with the equipment before you start real testing, we recommend that you go through the test examples described in this document using a simple adder as a DUT or alternatively your own DUT.

This example uses two generator channels, a control channel and an analyzer channel. The two generator channels are passively added and the resultant sum input into the analyzer channel. The input to the analyzer and the control channel are viewed on an oscilloscope.

Example 1 is a complete procedure, illustrating the necessary steps to generate and analyze a cell stream. This example should take approximately 30 minutes to read through and perform.

Example 2 shows how to construct your own cells from the library of segments. The cell created in this example contains an inserted error, which is detected by the analyzer and displayed as a bit error rate. This example should take approximately 10 minutes.

Example 3 shows how to use the external clock modes of the analyzer, in particular demonstrating Cell-Detect mode. This example should take approximately 20 minutes.

Conventions used in this guide

This Getting Started Guide uses certain notation for mouse usage. Table 1explains this notation:

Table 1 **Mouse Action Notation**

Instruction	Means
Click	Click the left mouse button
Double-click	Click the left mouse button twice rapidly

Buttons that can be clicked in the graphical user interface are depicted in this guide with a line above and a line below (for example, click $\overline{\text{OK}}$, means move the mouse pointer over the button marked “OK” and click the left mouse button).

The graphical user interface contains fields where values or text must be entered from the keyboard. If you click in these fields you get a cursor which you can use to move the edit point of the entry field. If you double-click a field, the entire current entry is highlighted and will be overwritten on the next key press.

This guide uses the term <Enter> to represent the ‘Enter’ key on your keyboard. This name is keyboard dependent and may also be labelled <Return>, <CR> and so on.

Example 1: Measuring Bit Error Rate (BER)

Purpose of Test

- To demonstrate the general procedure for transmitting and analyzing a cell stream using the HP E4859A

Key Points

- Starting and stopping the software user interface
- Connecting channels
- Setting up generator and analyzer voltage levels/threshold
- Setting up the analyzer to analyze a specific generator cell and segments
- Selecting a clock mode
- Setting up a trigger output
- Setting up bit rate, frame length, sequence and guard time
- Automatic adjustment of cell transfer delay between the generators and analyzer
- Viewing test results

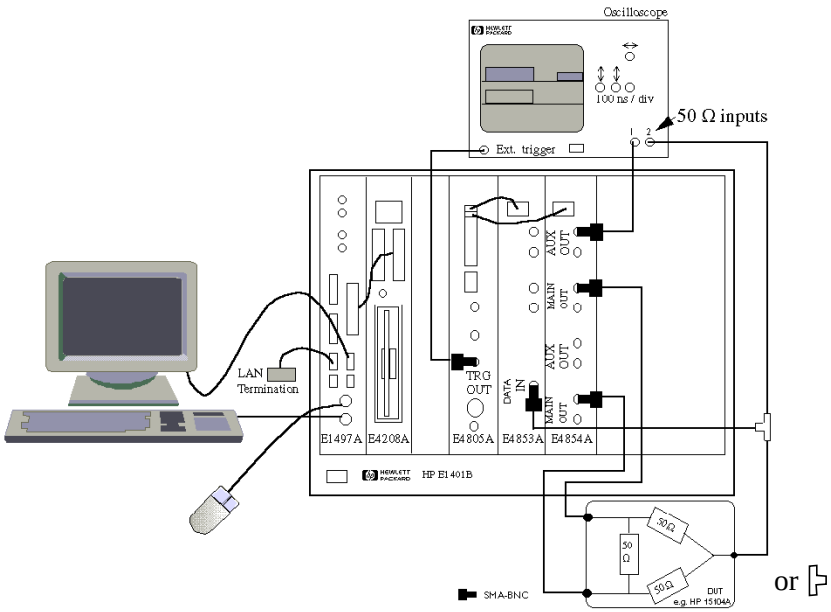
Test Setup

To perform the example you need:

- HP E4859A entry system (See Contents of Shipment in the Installation Guide HP E4859-91010). You need only 1 analyzer channel and 2 generator channels (1 HP E4854A and 1 HP E4853A module) to perform this example. Additional channels are not needed to demonstrate the system.
- Monitor, keyboard and mouse.
- 2/4 Channel Oscilloscope (to see the actual signal waveforms).
- DUT - this can be a simple adder as shown in the test setup diagram (for example, HP 15104A Adder), or if you prefer, your own DUT. It is also possible to use only a BNC T piece, but you should ensure that the analyzer input is 50 Ω (otherwise you will see large overshoots).
- (5 off) SMA(m)-BNC(f) Adapter connectors (1250-1200).
- (1 off) BNC T-Adapter (1250-0080).
- (7 off) BNC cables (8120-1839).

1 Connect up the system as shown in Figure 1.

Figure 1 Hardware Test Setup for Getting Started Examples 1 & 2



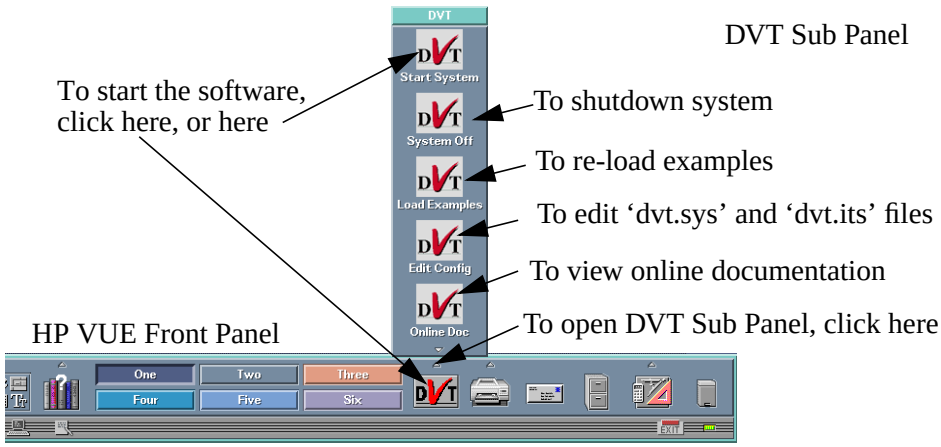
NOTE: The labels printed on the analyzer module front panel are dependent on the hardware installed. If option 002 is installed the data input channel is labelled IN 2 (Data). If option 002 is not present it is labelled DATA IN.

Test Procedure

Start the software

1 Start the software by clicking the **DVT** icon in the HP VUE Front Panel, alternatively you can open the DVT Sub Panel and click on the Start System icon.

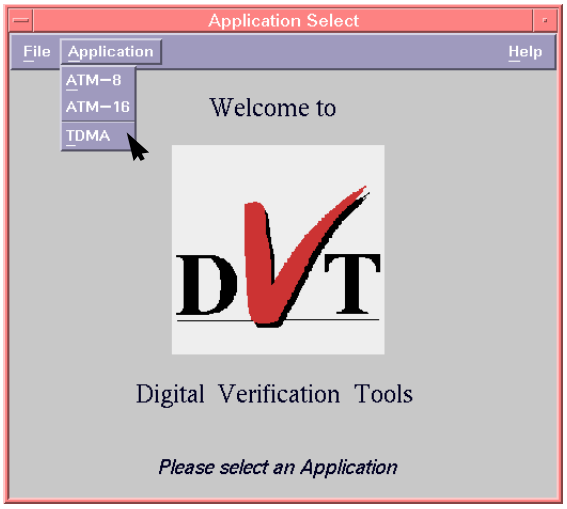
Figure 2 HP VUE Front Panel showing Start Software and System Off icons



Select your Application

2 After starting the software, select the TDMA application:

Figure 3 Application Select Window



After selecting the application the Main Connection window is displayed.

How to stop the software and shutdown the system

When you want to exit the software, use the 'File->Exit' menu option in the Main Connection window. Then before switching the system off, open the DVT Sub Panel and click the **System Off** icon, then wait for the power down instruction (see Figure 2).

WARNING: **If you do not follow this controlled shutdown procedure, you may corrupt the file system**

Other DVT Sub Panel Icons

This icon re-loads the set of examples from the file system into the database. This includes several demonstration settings and the example settings and cells described in this guide. These settings are already loaded if your system has been factory pre-installed.
Examples can only be loaded when the user interface software is NOT running. After clicking this icon you must wait for the message that the examples have been successfully loaded before starting the software (this takes about 30 seconds).

This opens a text editor containing the system files 'dvt.its', 'dvt.sys' and 'license.dat'. The 'dvt' files define the VXI address and communication mechanism for the modules contained in the frame, and define how these modules are grouped together to create a system. They are automatically configured when the system is first installed. If the software is started and it cannot find these files, then they are created automatically based on the current system hardware. For more information on these files, please refer to the on-line Help.
File 'license.dat' holds the licence keys for all installed applications. For more information on licensing refer to the installation guide.

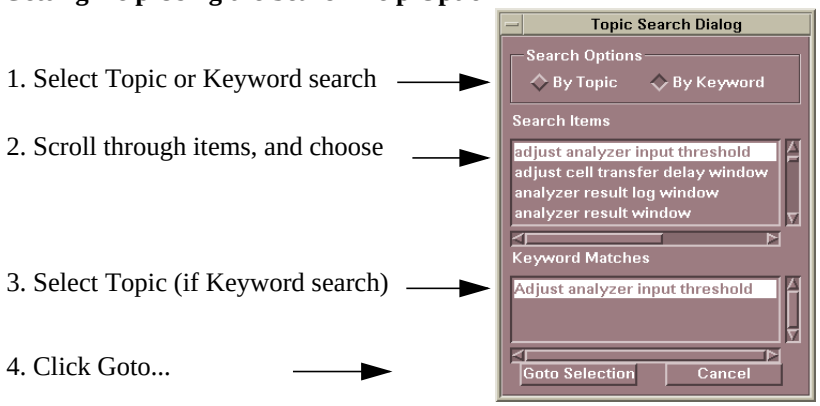
Getting Help

You can get help at any time when running the software. Simply press the <F1> key on your keyboard or alternatively click on the Help menu option at the top right hand corner of the displayed window.

For example, pressing the <F1> key in the Main Connection window opens a Help topic relating to this window. Throughout this Help topic are a list of related topics which you may also find helpful. In general you will find a help topic which explains the functionality of the window (explanations of entry fields, buttons and so on) and also related task oriented procedures.

You can also get help using the **Search** feature in the Help viewer. To open the Search dialog, click the Search button in the Help viewer:

Figure 4 Getting Help Using the Search Help Option

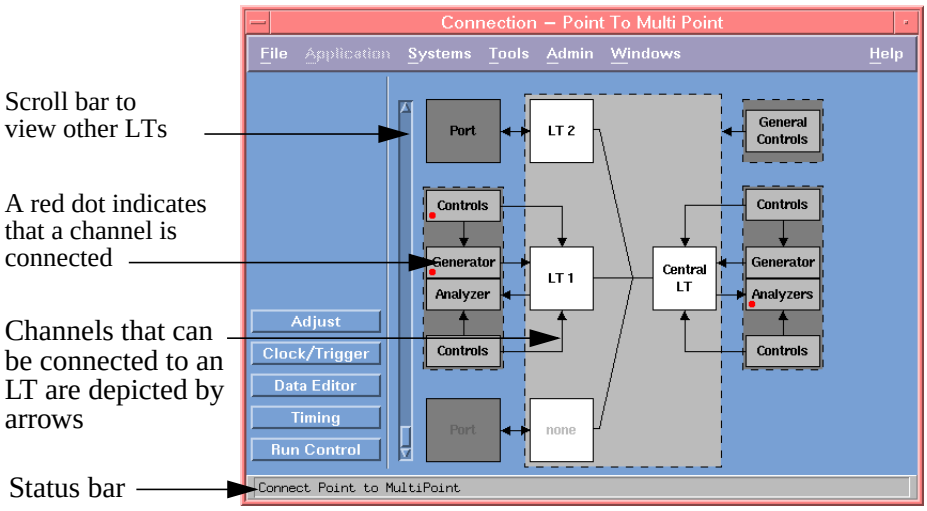


Main Connection Window

After starting the software the Main Connection window is displayed.

The graphic shows a Point to Multiple Point Network configuration with a single Central LT (Line Terminator) which is sometimes called the Central Office, connected through a network to several Subscriber LTs or Network Interfaces. Each Subscriber LT is connected to the Central LT by a ‘passive directional coupler’. This means that the bit rate on both sides of the coupler is always the same and that upstream transmissions from Subscriber LT are directed to the Central LT only. Downstream transmissions from the Central LT are broadcast to all Subscriber LTs.

Figure 5 The Main Connection Window



You will notice a scroll bar immediately to the left of the graphic, use this to display the other LTs, to which you can connect generator and analyzer channels.

The status bar at the bottom of the window gives help type information on the current cursor position in the window.

The light grey area in the center of the graphic represents the device under test (DUT).

Connecting Generator Channels

Connecting a channel means mapping a Generator, Analyzer or Control channel in the software user interface to a connector on a module front panel.

Each physical connector has a default name according to the frame the module is in, the slot number within the frame, and the module channel.

For example ‘Gen1M_5_1’ means:

Table 2 Default Naming Convention for Channels

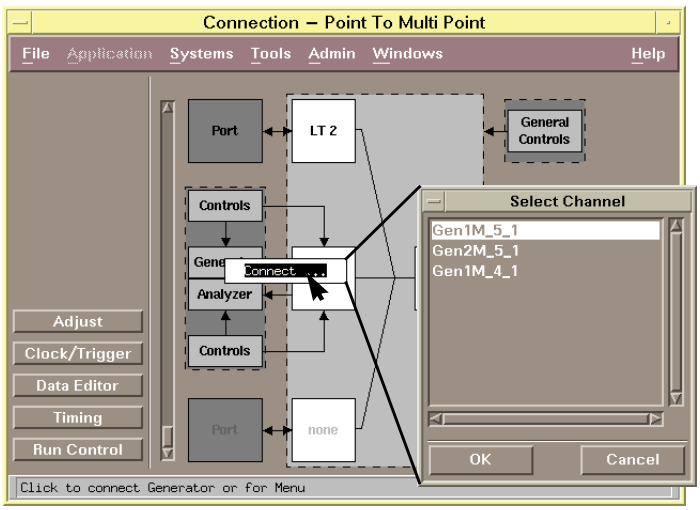
Part	Meaning
Gen	Generator channel (Ana for analyzer).
1M	Generator 1, MAIN output (1A means, Generator 1, AUX output).
5	Slot number 5 within the frame (slot numbers are engraved at the bottom and top of the frame).
1	Frame number 1 (for future expansion).

To connect a channel:

1 From the Main Connection window, center the display around LT1.

2 With the left mouse button, click on the Generator channel box that has an arrow pointing to LT1. Keeping the mouse button pressed, select the ‘Connect...’ option, then release the button.

Figure 6 To Connect a Generator Channel



3 The list of available (not already connected) channels is displayed. Select the channel ‘Gen1M_5_1’ and click OK.

NOTE: Depending on the amount of modules and the location of these modules within your frame, the amount and the names of the channels may be different to the ones shown in this guide.

You now have a channel connection between the main output of the dual generator module channel 1, and the Generator connected to LT1 in the user interface. You will notice a red dot is displayed in the Generator box.

After connecting a channel, that physical channel is removed from the list of available channels.

4 Using steps 1 to 3, connect generator channel ‘Gen2M_5_1’ to the Generator of LT2. You will need to use the scroll bar to display LT2, before connecting this channel.

Connecting the Analyzer Channel

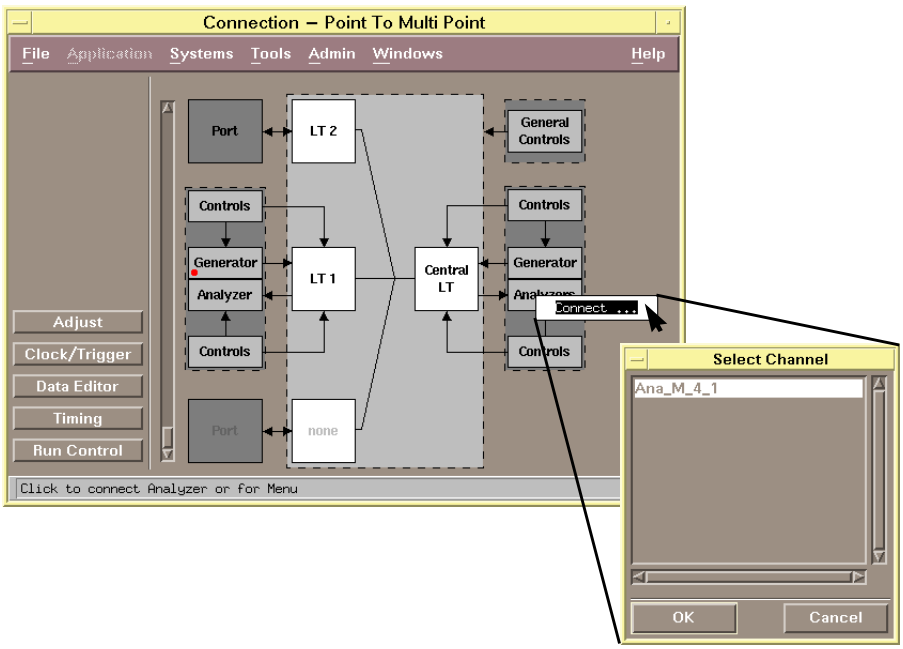
Connecting an analyzer channel means mapping the physical input on the HP E4853A front panel, to a user interface Analyzer channel.

An analyzer measures cells coming from **one** generator. Individual segments or groups of segments within this generator cell can be singled out for analysis. (how to do this is shown in a later section)

1 With the left mouse button, click on the Analyzer channel box that has an arrow pointing from the Central LT. Keeping the mouse button pressed, select the 'Connect' option, then release the button.

2 From the list of available analyzer channels (the list will depend on how many analyzers you have in your system), select the channel 'Ana_M_4_1' and click OK.

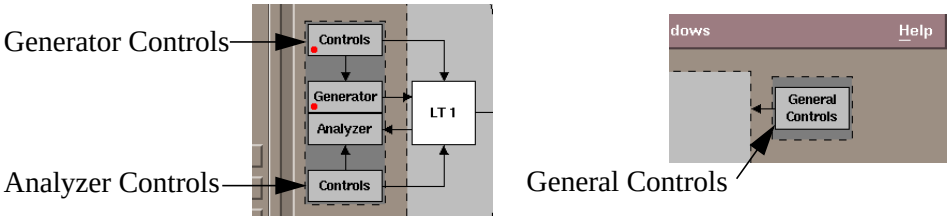
Figure 7 Connecting the Analyzer



NOTE: Depending on the amount of modules and the location of these modules within your frame, the amount and the names of the channels may be different to the ones shown in this guide.

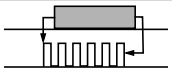
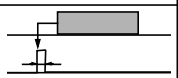
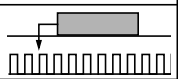
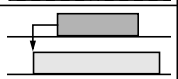
Control Channels (background information)

There are 3 types of control channels: Generator Controls, Analyzer Controls and General Controls. These are located in the Main Connection window by the word “Controls”:



A control channel provides an additional signal that can be related to the MAIN generator output cell (Generator Controls), start of frame (General Controls) or cells arriving at the analyzer INput (Analyzer Controls). AUXiliary or MAIN outputs may be used for outputting the control signal Control signals are depicted in the Timing window graphic in blue (see “Set Up Timing (Bit Rate and Frame Length)” on page 29). Depending on the control channel type it can be programmed to provide either envelope, bursted clock, reset pulse, continuous clock or user defined cell outputs:

Table 3 Generator Controls

Control Signal	Waveform	Parameters	MAIN Ranges	AUX Ranges
Envelope (MAIN and AUX)		Start before cell: Stop after cell:	0 to +/- 24 ms	0 to 24 ms
Bursted Clock (MAIN and AUX)		Start before cell: Stop after cell:	0 to +/- 24 ms	0 to 24 ms
Reset Pulse (MAIN only)		Start before cell: Width:	0 to +/- 24 ms	not available
Continuous Clock (MAIN and AUX)		Clock delay:	-1/2 to +1/2 of clock period	fixed to 0
User Defined Cell		Cell Start:	0 to +/- 24 ms	+/- 3 µs

If the MAIN Generator output is not connected then the corresponding AUX output cannot be used.

Analyzer Controls

For Analyzer Controls you may only connect MAIN outputs. User defined cells cannot be generated from analyzer controls. The control signal may be generated Once Per Analyzed Cell/ or once per Received Cell.

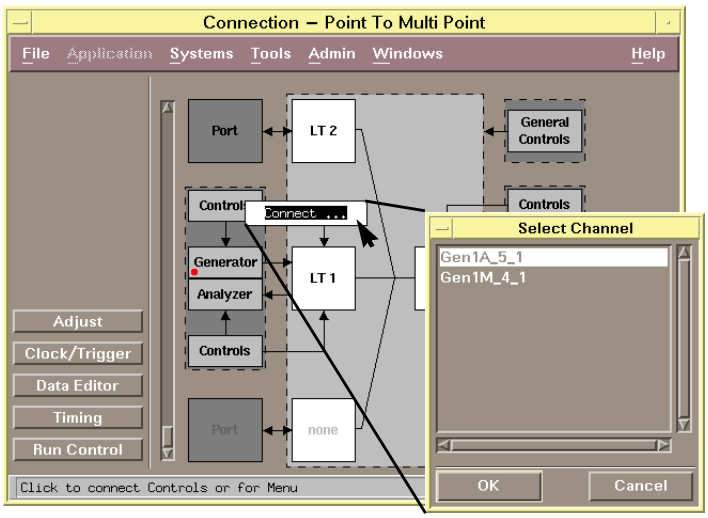
General Controls

For General Controls (the channel box in the top right corner of the Main Connection window) you may connect MAIN and AUX outputs and generate only user defined cells. General controls are generated relative to the start of frame.

Connect Generator Control Channels

1 With the left mouse button, click on the Controls channel box that has an arrow pointing to LT1 and an arrow from the connected Generator. Keeping the mouse button pressed, select the ‘Connect ..’ option, then release the button.

Figure 8 Connecting Control Channels



2 From the list of available channels, select the channel ‘Gen1A_5_1’ and click OK.

NOTE: Depending on the amount of modules and the location of these modules within your frame, the amount and the names of the channels may be different to the ones shown in this guide.

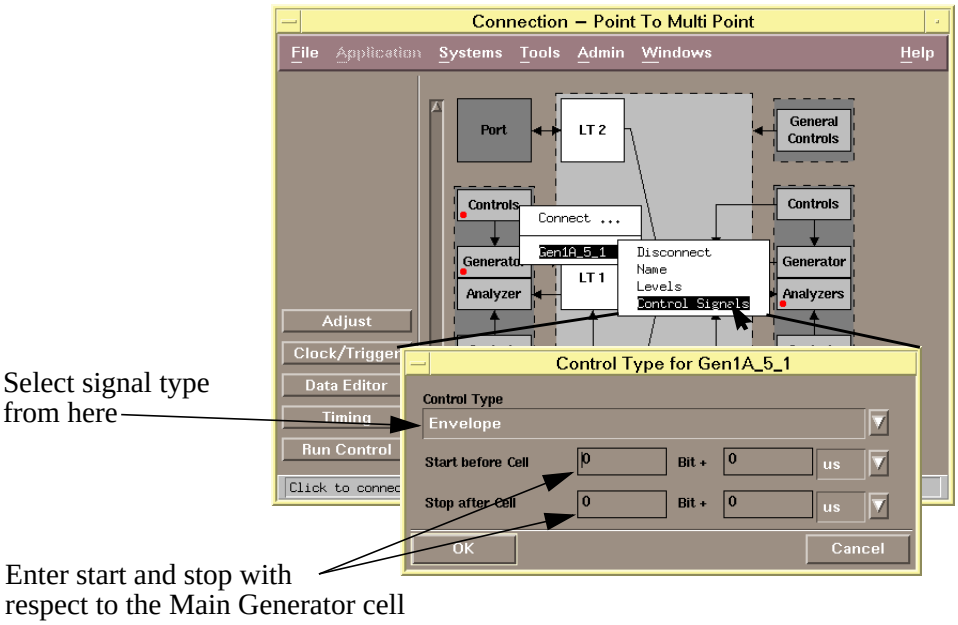
For our test example the auxiliary channel is physically connected to the scope.

To select a control signal type:

1 With the left mouse button, click on the connected Controls channel box. Keeping the mouse button pressed, select the '<channel name>->Control Signals' option, then release the button.

This opens the Control Type window:

Figure 9 Selecting a Control Signal Type



2 From this window you can select the control signal type, and start/stop relationship to the main output cell. For our example, we will leave the signal type as an 'Envelope', and 'Start before cell' and 'Stop after cell' as 0.

NOTE: The actual start and stop timing of the outputs are generated with a 1 single bit granularity. This means that if you enter start/stop timing values that are smaller than the bit period, they will be rounded to the nearest clock. For example, if using a 100 MHz bit rate (10 ns period), and you enter a 'Start before cell' of 17 ns, then the actual output will be generated 20 ns before the cell. However, the Timing window graphic will show the 17 ns.

3 Click Cancel to leave the Control Type window.

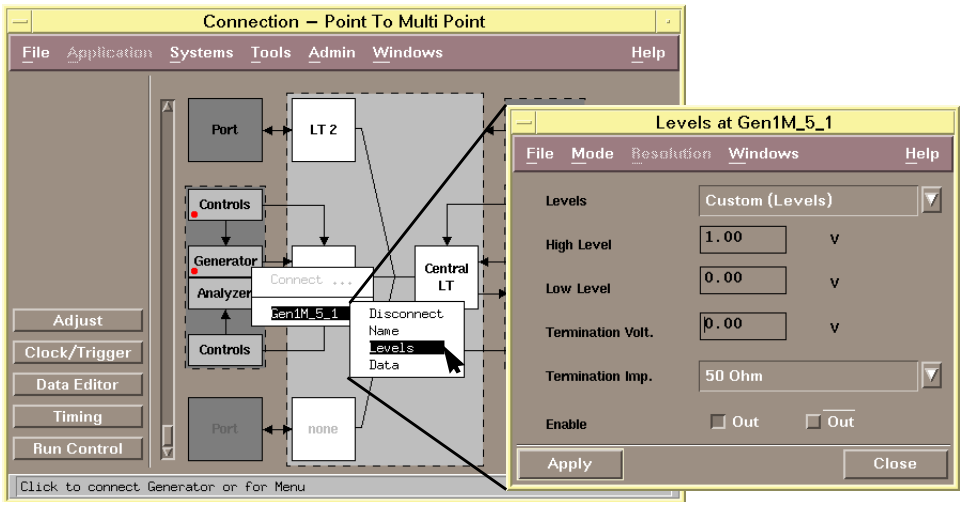
Set Up Voltage Levels for each Generator/Control Channel

For each of the outputs that we have connected (2 MAIN and 1 AUX) we must define the voltage levels and input impedances of the DUT.

For the DUT shown in Figure 1, each channel output drives into 50 Ω. We will set the high level output to be 1 Volt into 50 Ω. We need to do this for each individual output we are using:

1. From the Main Connection window, center the display around the LT unit that the channel is connected to. To display different LT units use the scroll bar to the left of the connection graphic.
2. With the left mouse button, click on the channel box for the channel you want to set. Keeping the mouse button pressed, select the '<channel name>->Levels' option and release the mouse button to open up the Voltage Levels window.

Figure 10 Setting Output Voltage Levels



3. Select 'Custom (Levels)'
4. Enter a value of 1.00 volt for the High Level, 0.00 volts for the Low Level and Termination Voltage, and 50 Ω Termination Impedance,
5. Enable the outputs by clicking on the 'Enable Out' option.
6. Click Apply to download the values to the module

   The indicator on the module front panel directly above the connector should light up immediately after clicking the Apply button.
7. Click Close to return to the Main Connection window.

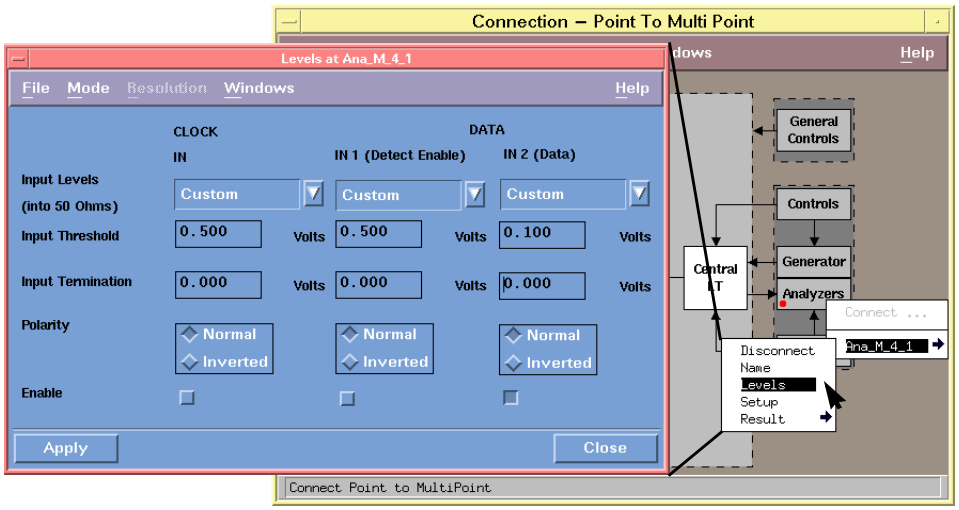
Repeat the above steps for the other Generator channel, and for the Control channel. For the other Generator channel enter a value of 0.75 volts for the High Level. This will make the channels easier to distinguish on the scope.

Set Up Voltage Levels for the Analyzer Channel

For each analyzer channel you want to use you must set up the input threshold and termination voltages.

1 With the left mouse button, click on the Analyzer that you connected to the Central LT. Keeping the mouse button pressed, select the '<channel name>-> Levels' option and release the mouse button to open up the Voltage Levels window.

Figure 11 Set Up Levels on an Analyzer Channel



NOTE: The appearance of the Analyzer levels window is dependent on the hardware installed. If option 002 is installed the data input channel is labelled IN 2 (Data). If option 002 is not present it is labelled DATA IN. These names correspond to the labels printed in the module front panel

2 Select 'Custom' for the Input Levels.

3 Enter a value of 0.10 volts for the Input Threshold, and 0.00 volts for the Input Termination Voltage.

4 Ensure 'Normal Polarity' is selected.

5 Enable DATA IN 2 (Data) by clicking on the 'Enable' option.

6 Click Apply to download the values to the module

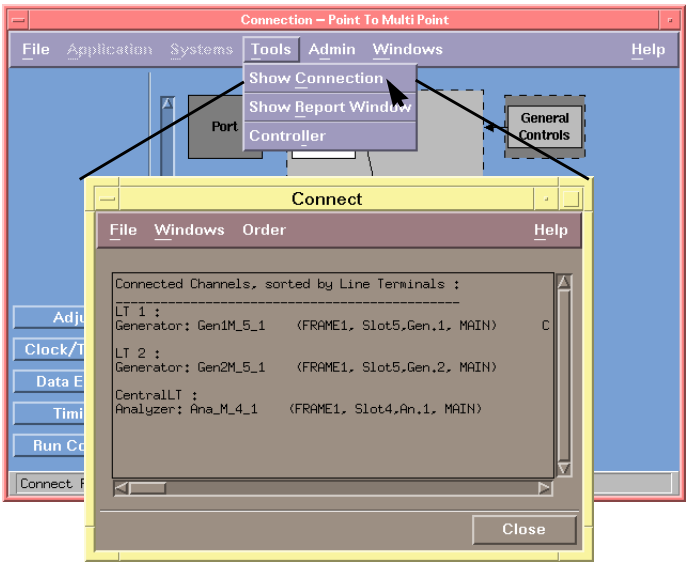
The indicator on the module front panel immediately above the connector should light up immediately after clicking the Apply button.

7 Click Close to return to the Main Connection window.

Displaying a Summary of Connections

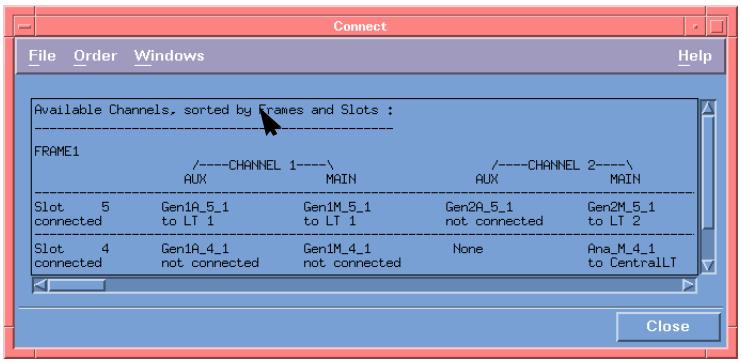
You can display a summary of connections at any time using the ‘Tools->Show Connection’ menu option:

Figure 12 Show Connection Window (ordered by LT)



The default view, orders the connections by LT unit. You can view the connections ordered by frame using the ‘Order->Frame’ menu option in the Show Connection window.

Figure 13 Show Connection Window (ordered by frame)



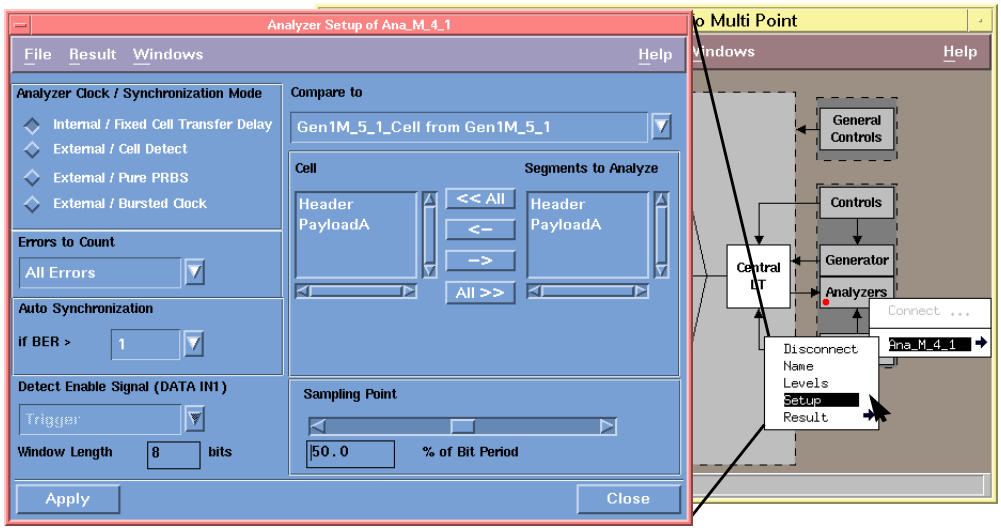
Setting Up the Analyzer

The Analyzer Setup window is used to define:

- source and type of sampling clock (synchronization mode)
- types of errors to count (errored ones, errored zeros or all errors)
- synchronization BER threshold
- which cell, from which generator to analyze
- which segments within that cell to analyze
- sampling point of the data with respect to the bit period

1 Open the Analyzer Setup window using the ‘Analyzer->Setup’ option (see Figure 14)

Figure 14 Analyzer Setup Window



2 Internal or system clock is used to sample data. Select the ‘Internal/Fixed Transfer Delay’ Clock/Synchronization mode.

3 Select the ‘All Errors’ option.

4 Disable BER synchronization by selecting ‘Auto synchronization if BER > 1’.

5 Select the ‘Gen1M_5_1_Cell’ cell and generator from the ‘Compare to’ field

Even though no cell was created and associated to this generator manually, the default cell containing an 8 bit header and 16 bit PRBS is automatically created and associated to a Generator channel when it is connected.

6 Ensure that the Sample Point is set to 50% of bit period. For more information on Sampling point see “Adjusting Analyzer Sampling Point During a Test” on page 37, or on-line Help.

7 Click Apply to download the values to the module, and then click Close.

Set System Clock Mode and Trigger Output Mode

From the Clock/Trigger Out window you can select the clock mode and specify the trigger output signal.

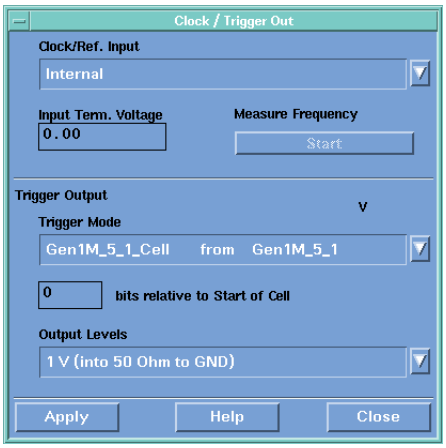
The following System Clock Modes are available:

Table 4 **Clock Modes**

Clock Mode	Description
Internal	The bit repetition rate is synthesized from an internal 10 MHz crystal source (default mode).
VXI Reference	The VXI backplane clock is used as a reference for generating the bit repetition rate. Use this mode when you want to clock the HP E485xA module(s) and other instruments in the same VXI frame from the 10 MHz backplane clock.
1,2,5,10 MHz External Reference	An external clock (1,2,5 or 10 MHz) is used via the CLOCK/REF INPUT on the clock module front panel and is used to synthesize the bit rate. Use this mode if you want to: - use a very accurate/stable external 1/2/5/10 MHz source - use the same 1/2/5/10 MHz source for multiple instruments
External Bit Rate	The CLOCK/REF INPUT on the clock module front panel is used to generate the bit rate at exactly the same frequency. Use this clock mode if you want to clock the HP E485xA module output channels from an external system (e.g. 40MHz from a computer motherboard or a standard 155.2 Mb/s source).

1 From the Main Connection window click on the Clock/Trigger button to open the Clock/Trigger Out window.

Figure 15 **Clock / Trigger Out Window**



2 Ensure the ‘Clock/Ref Input’ is set to ‘Internal’.

3 Select the ‘Trigger Mode’ as ‘Gen1M_5_1_Cell from Gen1M_5_1’

This generates an active low pulse of one bit period duration at the start of each cell from the specified generator. A Continuous Clock mode is also available.

4 Set the ‘Output Level’ to ‘1V (into 50 Ohm to GND)’

5 Click Apply to download the values to the clock module

6 Click Close to return to the Main Connection window.

Set Up Timing (Bit Rate and Frame Length)

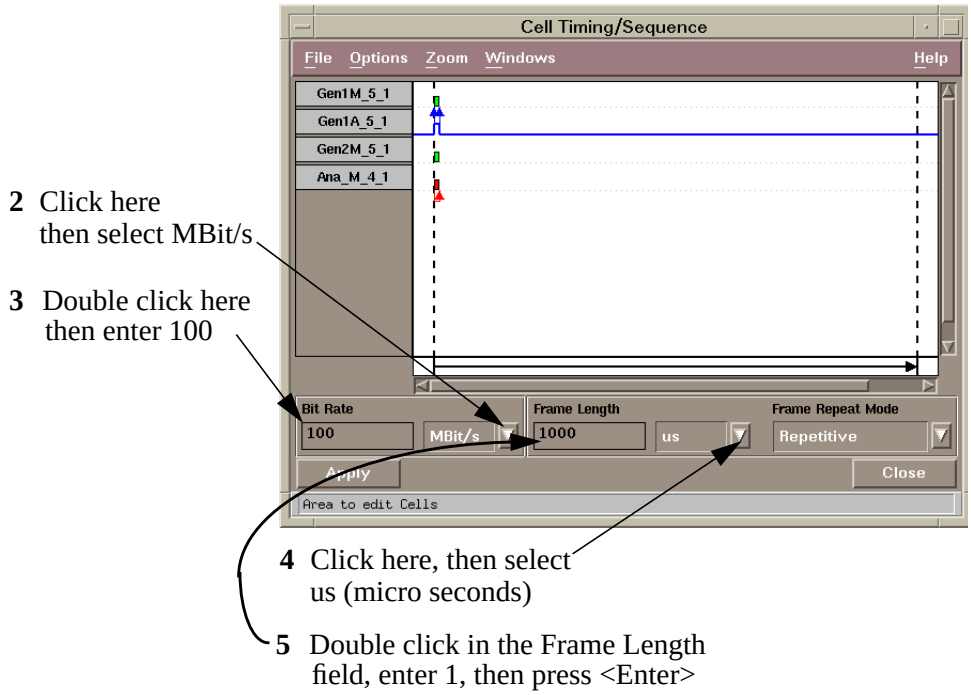
The Timing window is used to set up bit rate, frame length, guard time between cells and the sequence that cells are transmitted from different generators.

For this example you will set up a bit rate of 100 Mb/s, a frame length of 100 µs and a guard time between cells of 5 bits (5 µs).

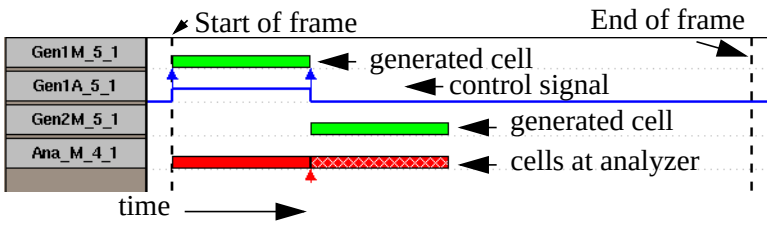
1 From the Main Connection window click on the Timing button to open the Cell Timing and Sequence window.

This window shows the lengths of the frame, cells and guard time proportional to one another.

Figure 16 Cell Timing and Sequence Window



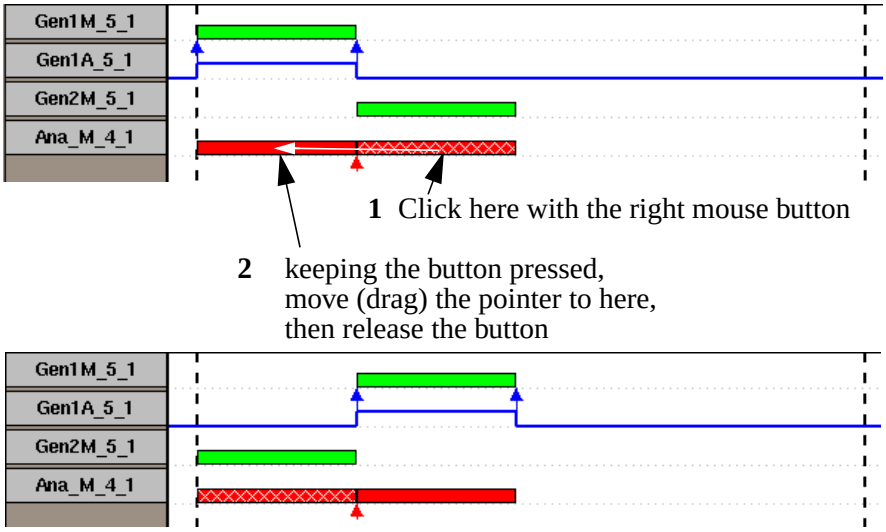
After setting up these timing parameters you will see the timing relationships between the cells transmitted by the generators (green) and received by the analyzer (directly below in red) within the frame (dashed vertical lines). Cells that are analyzed are shown in solid red, and cells that are not analyzed are hatched or shaded. The control signal is shown in blue directly below the associated cell:



Set Up Timing (Cell Sequence or transmission order)

The cell sequence defines what order the cells are transmitted. The cell sequence is changed directly from the Cell Timing / Sequence window:

Figure 17 Changing the Cell Sequence



The timing graphic now shows that the cell generated by ‘Gen2M_5_1’ is transmitted before the cell generated by ‘Gen1M_5_1’.

NOTE: Depending on the amount of modules and the location of these modules within your frame, the amount and the names of the channels may be different to the ones shown in this guide.

The general procedure for changing the cell sequence is:

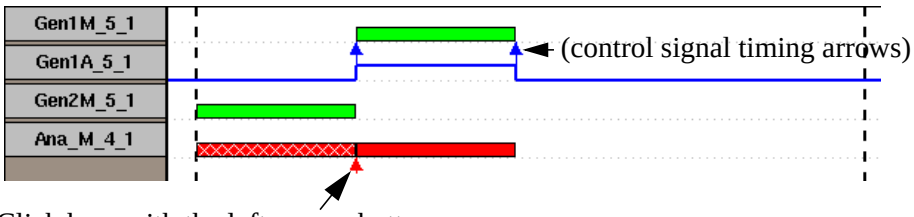
1. Click and hold the right mouse button on the analyzer (red) cell you want to move.
2. Keeping the mouse button pressed, move the mouse over the analyzer cell where the selected cell is to be placed (the moved cell is always placed in front of the cell where the mouse button is released).
3. Then release the mouse button.

When you move a cell its guard time moves with it. If you move a cell to the first position its guard time is automatically set to zero.

Set Up Timing (Guard Time)

The guard time is also modified from the Cell Timing and Sequence window:

Figure 18 Setting Up Guard Time



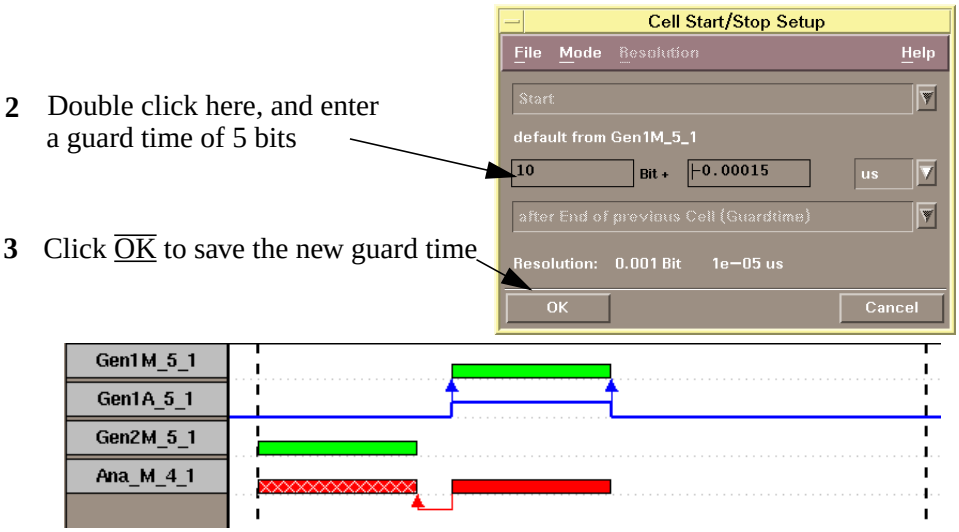
1 Click here with the left mouse button to open the Cell Start/Stop Setup window

(Alternatively, you can click on the analyzer cell (red), and select “Start/Stop Event”).

The Cell Start/Stop Setup window is used to define when a cell is started relative to the previous cell (this is defined as cell guard time).

NOTE: To change the start/stop timing for the control signal from the Timing window, you can click on the blue control signal timing arrows (see Figure 18)

Figure 19 Cell Start/Stop Setup Window



2 Double click here, and enter a guard time of 5 bits

3 Click OK to save the new guard time

The timing graphic now shows the guard time relative to the cell and frame lengths

4 Click Apply in the Timing window to download all the timing values to the modules.

5 Click Close to return to the Main Connection window.

Adjust Cell Transfer Delays Between Generators and Analyzer

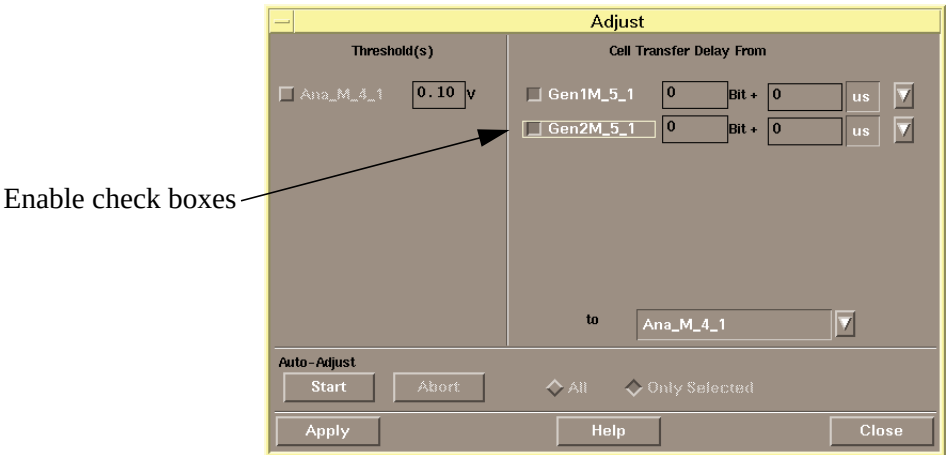
The cell transfer delay is the propagation delay between generator and analyzer, through the DUT. These values can be entered manually by keyboard entry or calculated automatically by auto-adjust.

The Adjust window is used for setting the cell transfer delay and analyzer input voltage thresholds. The analyzer threshold displayed in the Adjust window is the same as that entered in the Analyzer->Levels display.

To perform an automatic calculation of cell transfer delay:

1 From the Main Connection window click on the Adjust button. This opens the Adjust window:

Figure 20 Adjust Window



NOTE: Depending on the amount of modules and the location of these modules within your frame, the amount and the names of the channels may be different to the ones shown in this guide.

2 Enable the auto-adjust calculation by clicking the check boxes next to generators ‘Gen1M_5_1’ and ‘Gen2M_5_1’.

Hint: Make sure that the initial values are less than the expected delay. During the auto-adjust, any connected analyzer control signal used is generated at the initial transfer delay (the default is 0).

3 Click Start to start the auto-adjust of cell transfer delay.

4 After approximately 1 minute, the calculated cell transfer delays will be entered automatically into the corresponding entry fields.

5 Click Apply to download the values to the module, then click Close.

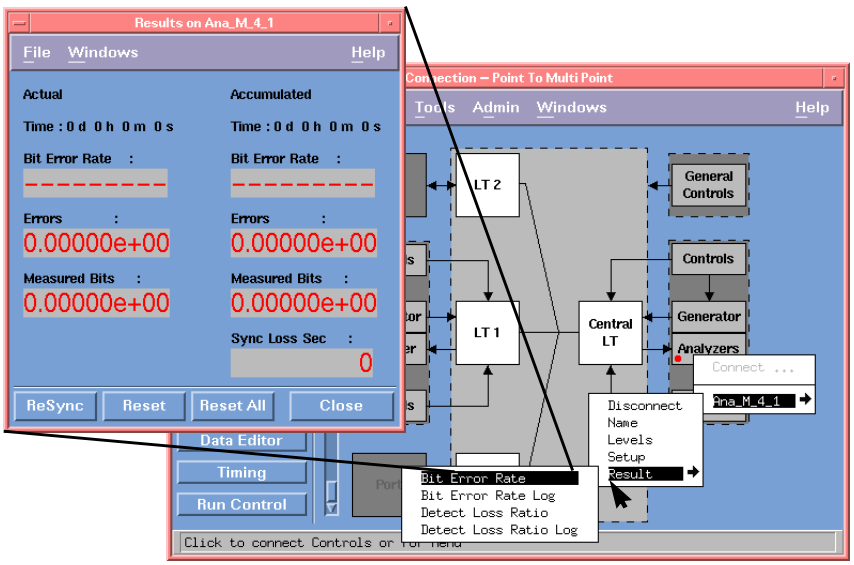
Open Results Window

There is one Bit Error Rate Result and one Bit Error Rate Log window for each analyzer that is connected in the Main Connection window.

The BER Result window displays the bit error rate, total number of error bits received and the total measured bits for both the last second and for Accumulated Measurement Intervals. The Measurement Interval is defined as a time period over which measurements are made, and is set up in the Run Control window.

1 Open the Analyzer Setup window:

Figure 21 BER Results Window



The ‘Actual’ displays on the left side of the window reflect the error values over the last second, that is for all modes (see Table 5, “Run Time Measurement Modes,” on page 34). In ‘Repetitive Interval’ mode the values over the last Measurement Interval are written to the Result Log.

The ‘Accumulated’ displays on the right side of the window display the accumulated values over all the elapsed Measurement Intervals. These values are updated at the end of every Measurement Interval and are applicable in Repetitive Interval mode only.

The ReSync button initiates an analyzer synchronization on pure PRBS patterns.

The Reset button resets the counter values associated with this Analyzer Result window. All the entries in the Log window are cleared.

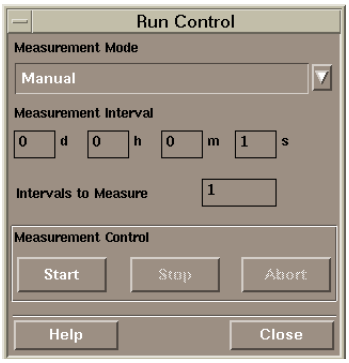
The Reset All button resets the counter values associated with all Analyzer Result windows simultaneously. All the entries in all Log windows are also cleared.

Run the test

Starting and stopping the generators and analyzer is performed from the Run Control window.

1 From the Main Connection window click on the Run Control button.

Figure 22 Run Control Window



As well as starting and stopping the test, you can use the Run Control window to set up the Measurement Mode. This defines how the measurement is run and affects the presentation of results:

Table 5 Run Time Measurement Modes

Measurement Mode	Description
Manual	A single measurement is performed which lasts until the Stop button is clicked. The results of the measurement can be viewed in the Results window of a connected analyzer.
Single Interval	One measurement is performed and lasts for the Measurement Interval. After the Measurement Interval the measurement stops automatically.
Repetitive Intervals (manual stop) and (loop)	Continuous back to back measurements are performed each lasting for the Measurement Interval. For each elapsed Measurement Interval a line is entered into the Results Log for each connected analyzer. This gives a complete history since the start of the measurement. If you use 'loop', a fixed number (defined by the field 'Intervals to Measure') of measurement Intervals are executed, then the measurement stops automatically.

2 Select the Measurement Mode 'Repetitive Intervals (manual stop)'.

3 Enter a Measurement Interval of 5 seconds (double click in the rightmost box of the Measurement Interval and enter '5' from the keyboard)

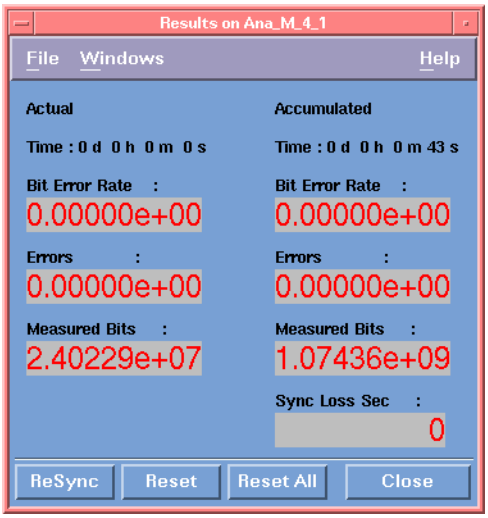
4 Start the measurement by clicking the Start button.

Verify results

Bit Error Rate Window

Typical results are shown in Figure 23 and Figure 24. With everything set up correctly the ‘Actual’ and ‘Accumulated’ bit error rates and Sync Loss Sec. should be 0.

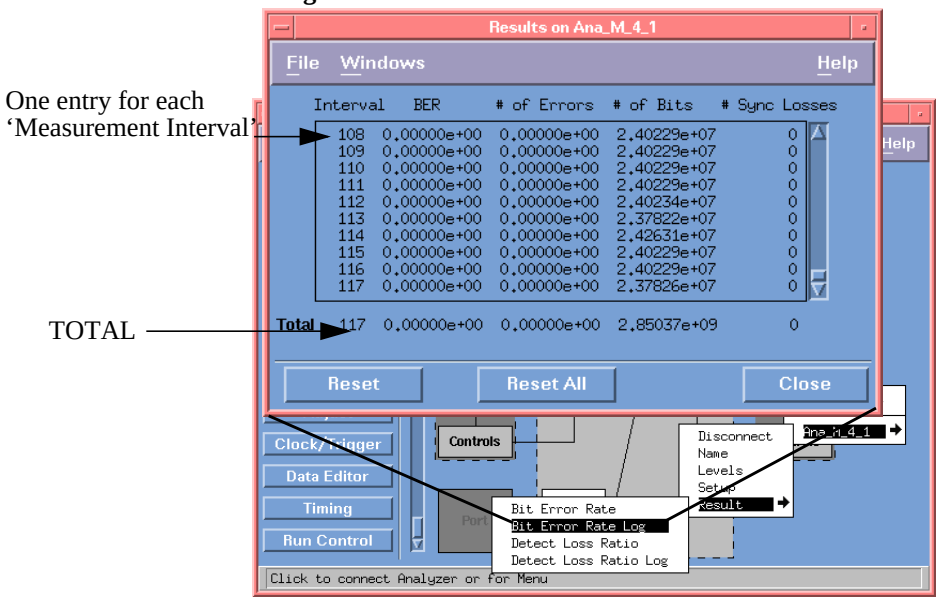
Figure 23 Bit Error Rate Results Window



Bit Error Rate Log Window

The Results Log window shown in Figure 24. It is opened in the same way as the Results window except BER Log is selected. This displays a complete history of each Measurement Interval since the start of the measurement.

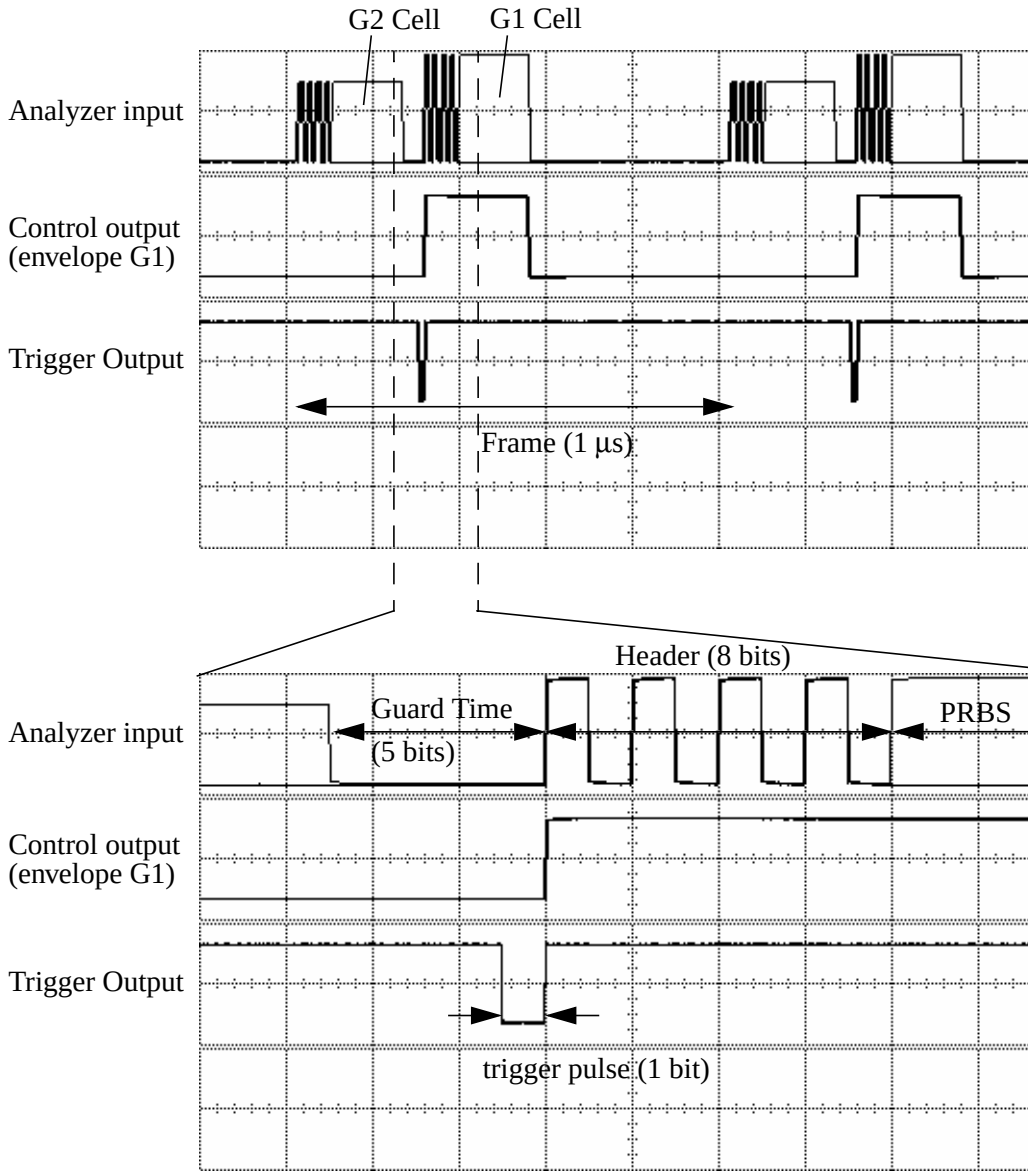
Figure 24 Bit Error Rate Log Window



Oscilloscope Traces

Typical analyzer traces are shown in Figure 25.

Figure 25 Oscilloscope Traces



Analyzer input - 200 mV/div, 50 Ω

Control Output and Trigger Output - 800 mV/div, 50 Ω

Top trace timebase is 200 ns/div, bottom trace is 20 ns/div.

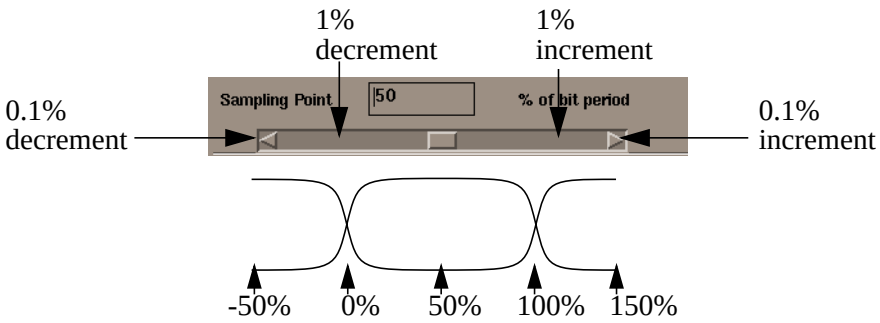
Adjusting Analyzer Sampling Point During a Test

You can adjust the analyzer sample point while a test is running:

1 Open the Analyzer Setup window (see “Setting Up the Analyzer” on page 27)

2 Adjust the sampling point using the horizontal scroll bar (see Figure 26)

Figure 26 Adjusting the Sampling Point of the Analyzer



You can adjust the sampling point between -50% and 150% of the bit period, where 0% is the start of the bit and 100% marks the end of the bit.s

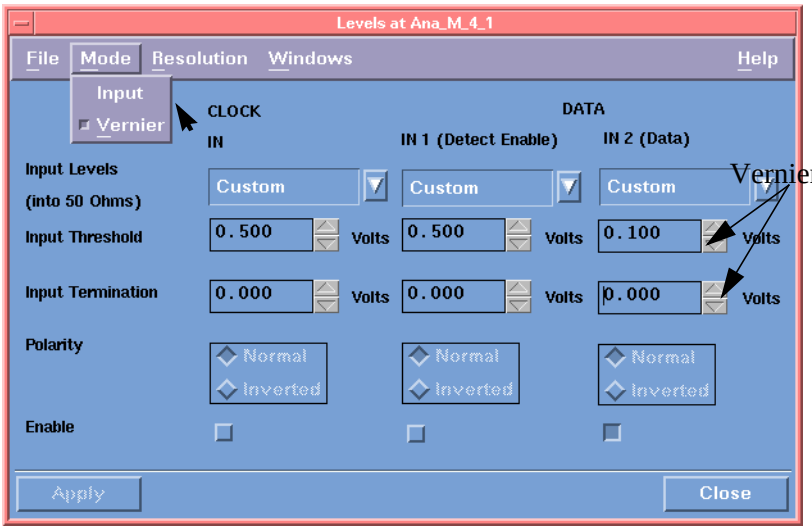
Table 6 Adjustment Limitations in internal clock mode

Bit Rate	Allowable adjustment
Bit Rate > 651 kHz	-50% - 150%
325,5 kHz < Bit Rate < 651 kHz	0 - 100%
Bit Rate < 325,5 kHz	0

Adjusting Analyzer Threshold/Generator Output Level During a Test

You can adjust the analyzer threshold and generator output levels while a test is running:

1. Open the appropriate Levels window (see “Set Up Voltage Levels for the Analyzer Channel” on page 25 or see “Setting Output Voltage Levels” on page 24).
2. From the Mode menu select Vernier



3. From the Resolution menu, select the increment/decrement step.
4. By clicking on the vernier buttons, you can adjust the values by a single resolution step per click, while a measurement is running.

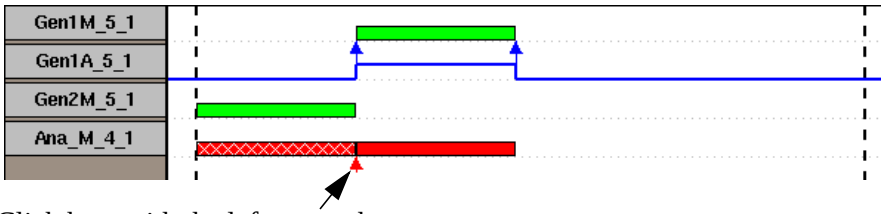
NOTE: You can also change the levels in ‘Input Mode’, simply enter the values from the keyboard and then click Apply.

Adjusting Cell Guard Times During a Test

You can adjust the guard time between cells while a test is running:

1 Open the Timing window (click on the Timing button in the Main Connection window).

2 Click on the guard time arrow (red) that you want to adjust:



Click here with the left mouse button
to open the Cell Start/Stop Setup window

3 From the Mode menu select Vernier

4 From the Resolution menu, select the increment/decrement step.

5 By clicking on the vernier buttons, you can adjust the values by a single resolution step per click, while a measurement is running.

NOTE: You can also change the Start/Stop times in 'Input Mode', simply enter the values from the keyboard and then click Apply.

NOTE: You may adjust the guard time of each cell by up to +/- 1 bit period while a test is running (see table below). If you need to adjust by more than this amount, then you need to stop the test first, adjust, Apply, and then re-start the test.

Table 7 Adjustment Limitations while a Test is Running

Bit Rate	Data Period	Allowable adjustment during a test
Bit Rate > 666,667 kHz	Data Period < 1.5×10^{-6}	+/- 1 bit period
333,333 kHz < Bit Rate < 666,667 kHz	1.5×10^{-6} < Data Period < 3.0×10^{-6}	+/- 0.5 of a bit period
Bit Rate < 333,333 kHz	Data Period > 3.0×10^{-6}	0

NOTE: It is also important to note that it is the start time of the cell relative to the start of frame that is limited. This means that if you adjust the guard time of the second cell in the frame by +1 bit, all subsequent cells are also adjusted by 1 bit period, and therefore additional delay may not be added.

NOTE: The Start/Stop timing events for user defined control cells (Generator or General Controls) may also be changed while a test is running.

To Stop the Test

To stop the test click Stop or Abort in the Run Control window. The Stop button stops the measurement at the end of the next Measurement Interval, whereas Abort stops the measurement immediately.

Saving the Setting to the Database

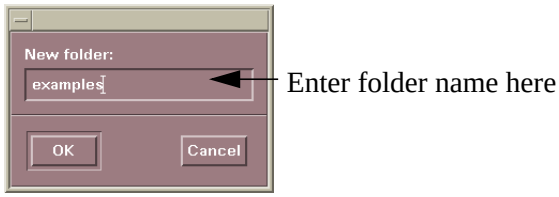
Settings are saved and opened from the database using the File menu in the Main Connection window. Folders allow you to save your settings within the database in a structured way like directories in a filesystem.

Create a Folder

1 Choose the ‘File->Save Setting as ...’ option from the Main Connection window.

2 Click New Folder, and enter the name ‘examples’ from the keyboard

Figure 27 Creating a New Folder

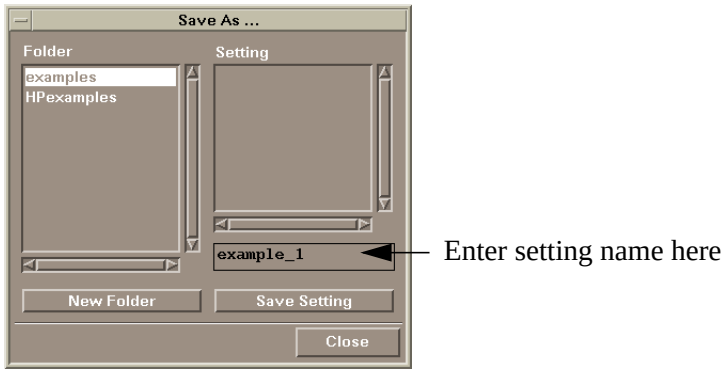


3 Click OK. You have now created a folder to save the setting into.

Save the Setting in the Folder

4 Click in the Setting box with the left mouse button, and enter the new setting name ‘example_1’ from the keyboard.

Figure 28 Saving the Setting



5 Click Save Setting to store this setting in the database.

Example 2: Using the Cell Editor to Create Cells

Purpose of Test

- To demonstrate how to create customized cells using the HP E4859A.

Key Points

- Creating a new cell.
- Adding segments to a cell.
- Inserting memory (pattern) segments.
- Inserting a PRBS with errors.
- Ordering the segments.

Test Procedure

This procedure continues from the system setting created in example 1. You can either continue directly from example 1 (or load the setting you saved in example 1), or you can load the delivered example 1 setting from the database (HPexamples folder, ‘gsm_example_1’ setting).

To load a setting from the database, use the File->Open Setting menu option in the Main Connection window. Select the folder, then the setting within the folder, and click Load Setting.

NOTE: After loading a setting from the database, it is necessary to click the Apply button in the Timing window before starting the test.

Creating a cell

In this example we will create a cell consisting of 3 segments - header, PRBS, and trailer. The header and trailer segments are memory based segments, which means we can edit the values within the segment. The PRBS segment is automatically generated, however we must select the type of PRBS and select the PRBS sequence length (how long the PRBS is, before it repeats its sequence). A PRBS can be one of the following types:

Table 8 **PRBS Types**

PRBS Type	Description
PRBS (Pure)	This PRBS is generated by the module hardware.
PRBS (1/0 Substitution)	The output data stream contains a string of 1's or 0's of programmable length.
PRBS (Var Mark Density)	The output data stream contains a programmed ratio of 1s to 0s.
PRBS (Error Insertion)	The output data stream contains a programmed number of error bits that will be detected by the analyzer. This is useful to verify that you are analyzing cells coming from a specific generator.

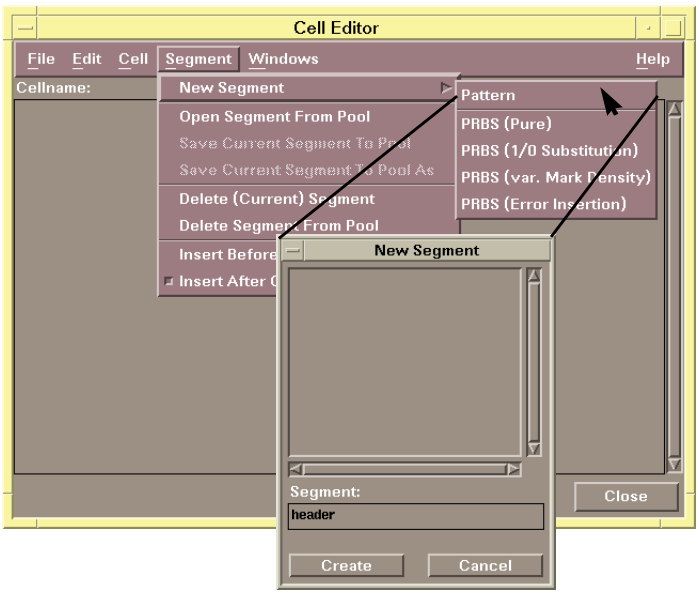
We will create a PRBS with a single error bit inserted. This is useful to verify that you are analyzing the correct source, as it represents a known Bit Error Rate.

1. Click on the Data Editor button in the Main Connection window. This opens the Cell Editor.
2. Create a new empty cell by selecting the ‘New Cell’ option from the Cell menu. This clears the current cell from memory.

Creating the Header

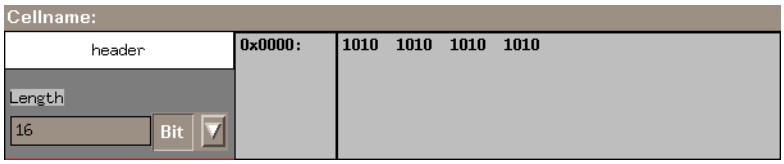
1 Create a new ‘Pattern’ type segment, by selecting the ‘New Segment->Pattern’ option from the Segment menu:

Figure 29 Cell Editor



2 Click in the ‘Segment’ entry field and type in the segment name ‘header’ from the keyboard. Click Create when done. The segment template is created in the Cell Editor window:

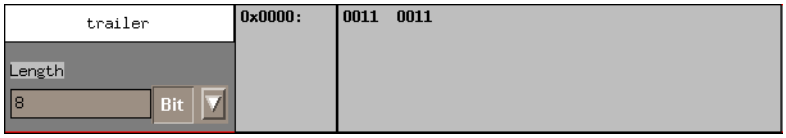
3 Click in the ‘Length’ field and enter the value 16 and then press <Enter>. Immediately after pressing <Enter> the pattern is displayed:



4 Click inside the pattern, and enter the bit stream ‘1010 1010 1010 1010’ from the keyboard. The header is now complete.

Creating the Trailer

1 Create the trailer exactly the same way as you created the header. Create the trailer with a length of 8 bits, and a value of '0011 0011':



Creating the PRBS Payload

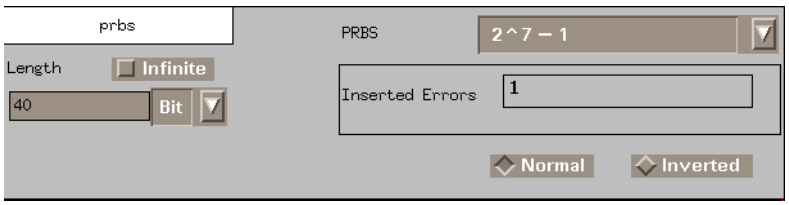
1 Create a new PRBS type segment, by selecting the 'New Segment->PRBS (Error Insertion)' option from the Segment menu.

2 Enter the name 'prbs' and click Create.

3 Enter a segment Length of 40 bits.

4 Select the PRBS '2^7 - 1'. This defines the length of the PRBS sequence.

5 Set 'Inserted Errors' to 1. This produces 1 error in the complete PRBS sequence (1 in 2^7-1, or 1 in 127):



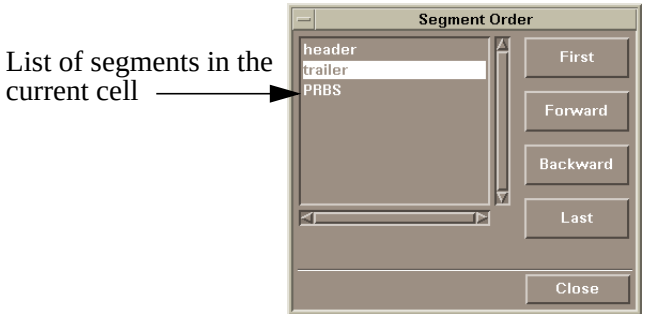
The 'Infinite' Length option is used for creating a continuous PRBS signal (please refer to the on-line Help, for how to create a Continuous signal).

Order the Segments

Because we want the trailer to come after the PRBS payload, we will re-order the segments.

1 From the Edit menu select the 'Segment Order' option. This opens the Segment Order window:

Figure 30 Segment Order Window



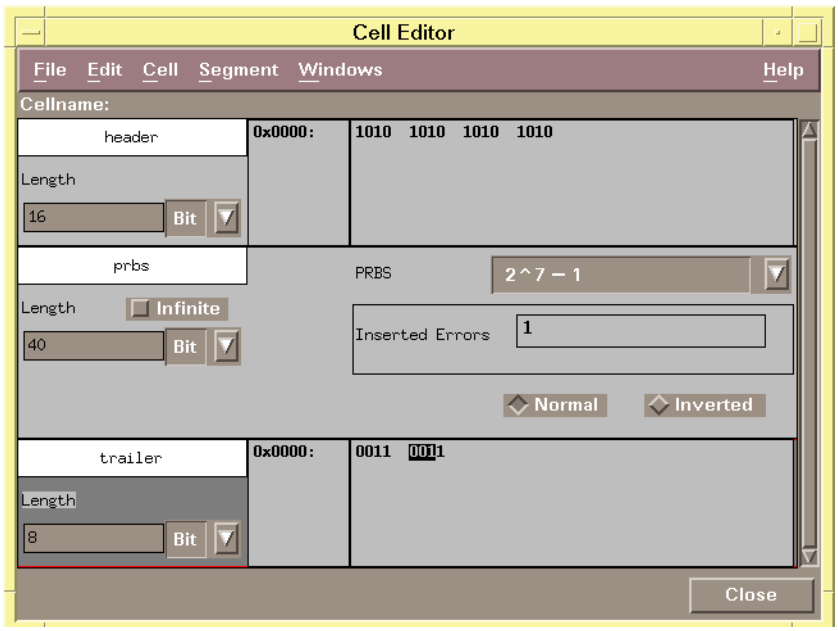
2 Click on the 'trailer' segment in the Segment Order window to that the segment is highlighted. The buttons on the right of the window are automatically enabled.

3 Click Last, to move the segment to the end of the cell.

4 Click Close to return to the Cell Editor.

The order of the segments should now be: header, prbs and then trailer:

Figure 31 Cell Editor Showing Complete Cell

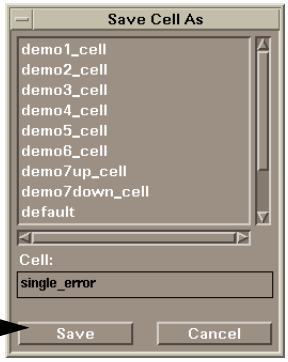


Saving the cell

1 From the Cell menu, select ‘Save Cell’.

2 From the dialog, click in the ‘Cell’ entry field and enter the cell name ‘single_error’ from the keyboard:

Figure 32 Save Cell Window



3 Click Save.

4 Click Close, to close the Cell Editor window.

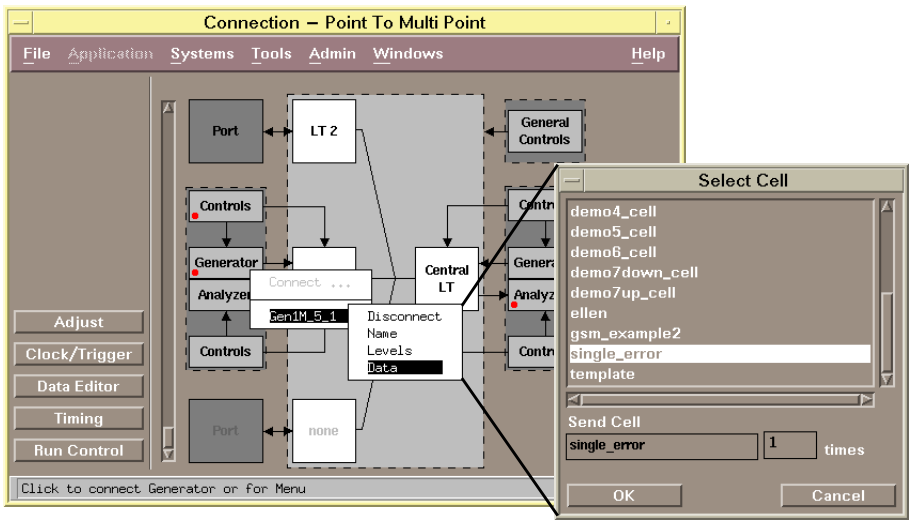
Associate the Cell to the Generator

In order to transmit this cell, you must attach it to a generator:

1 From the Main Connection window, center the display around LT1.

2 With the left mouse button, click on the Generator channel box that has an arrow pointing to LT1. Keeping the mouse button pressed, select the ‘Gen1M_5_1->Data’ option, then release the button.

Figure 33 Associating a Cell to a Generator



For generating multiple cells from the same generator use the “times” field in the Select Cell window.

3 Select the ‘single_error’ cell, that we just created.

4 Click OK.

Run the test

1 From the Main Connection window click on the Run Control button.

2 Set the Measurement Mode to ‘Manual’.

3 Click the Start button.

Verify results

Bit Error Rate Window

Because we have injected a single error in the PRBS payload we can verify that we are analyzing the correct generator.

Open the Results Bit Error Rate window:

1 With the left mouse button, click on the Analyzer channel box. Keeping the mouse button pressed, select the 'Ana_M_4_1->Results->Bit Error Rate' option, then release the button. This is an alternative method of opening the Result windows.

2 Verify that the Bit Error Rate reads 4.92126e-03. This can be calculated as follows:

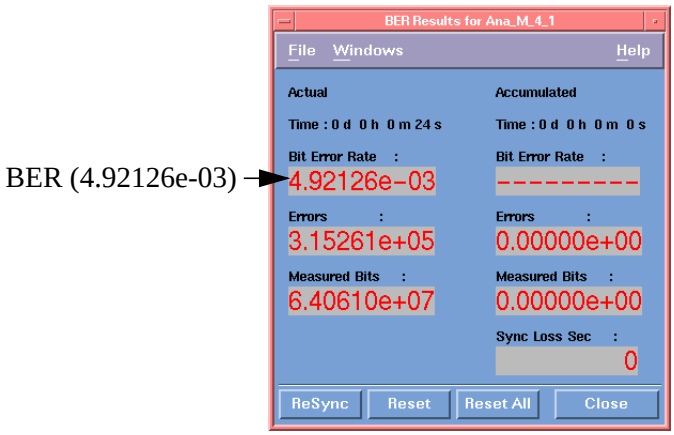
n = 1 (number of inserted errors in sequence length)
PRBS sequence length = $2^7 - 1 = 127$ (1 error in 127 bits)
Length of PRBS segment = 40 bits
Length of cell = 64 bits

$$BER = \frac{\text{Length of PRBS segment}}{\text{Length of cell}} \times \frac{n}{\text{PRBS sequence length}}$$

$$= \frac{40}{64} \times \frac{1}{127}$$

$$= 0.00492126$$

Figure 34 BER Results for Example 2



Save the Setting

Using the 'File->Save Setting as ...' option in the Main Connection window, save the setting as 'example_2' in the 'examples' folder (see "Saving the Setting to the Database" on page 40).

Example 3: Using External Analyzer Clock modes

NOTE: This example requires an analyzer with option 002 (external clock input).

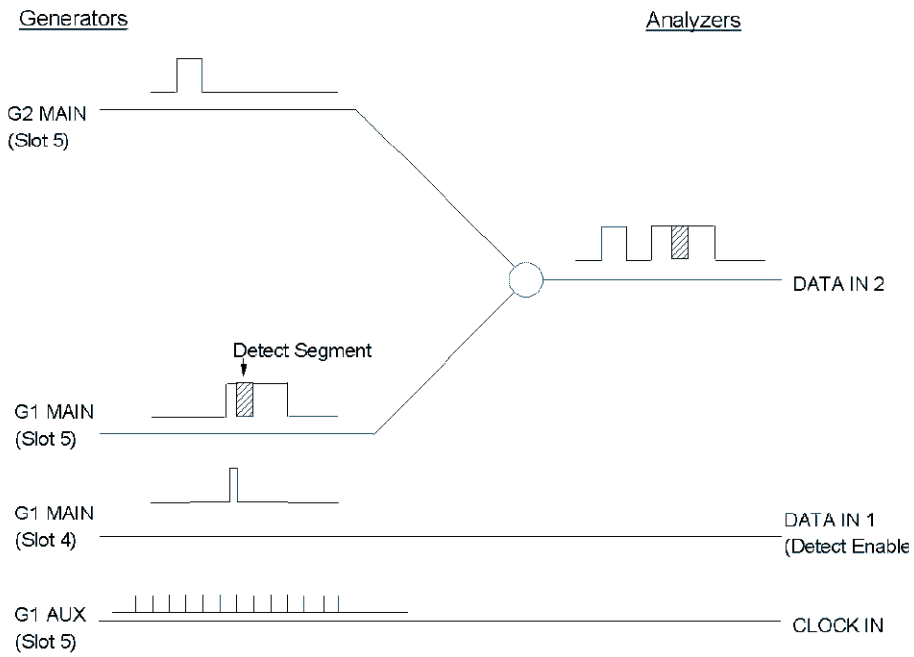
Purpose of Test

- To demonstrate the external clock analyzer modes of the HP E4859A

Key Points

- Creating a cell with a detect segment
- Setting up control channels to generate a clock input and a detect enable signal
- Setting up analyzer voltage levels/thresholds
- Setting up the analyzer to use the clock input in cell detect mode.
- Viewing test results and observing the effect of changing the sampling point and

Figure 35 Summary of Example 3 test setup.



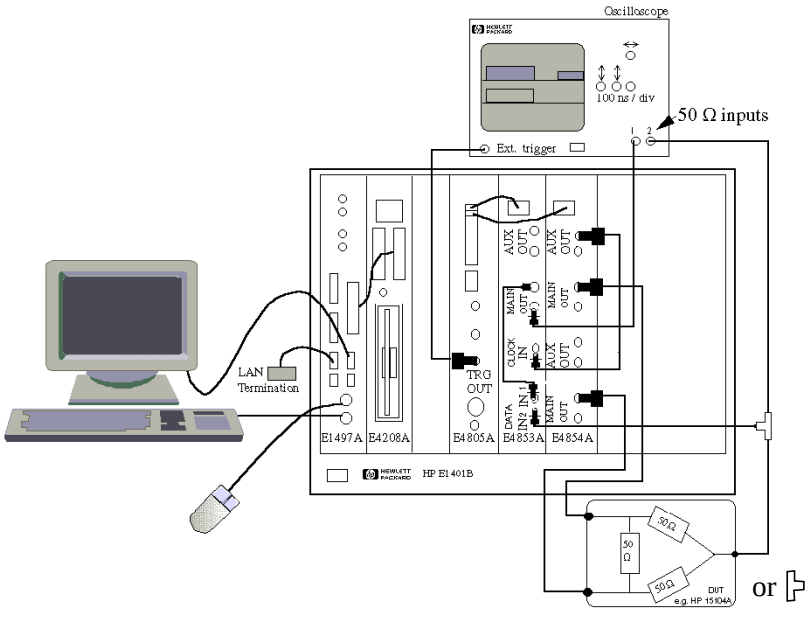
Test Setup

To perform the examples you need:

- HP E4859A entry system (See Contents of Shipment in the Installation Guide HP E4859-91010). You need only 1 analyzer channel and 2 generator channels (1 HP E4854A and 1 HP E4853A module) to perform these examples. Additional channels are not needed to demonstrate the system.
- Monitor, keyboard and mouse.
- 2/4 Channel Oscilloscope (to see the actual signal waveforms).
- DUT - this can be a simple adder as shown in the test setup diagram (for example, HP 15104A Adder), or if you prefer, your own DUT. It is also possible to use only a BNC T piece, but you should ensure that the analyzer input is 50 Ω (otherwise you will see large overshoots).
- (5 off) SMA(m)-BNC(f) Adapter connectors (1250-1200).
- (1 off) BNC T-Adapter (1250-0080).
- (7 off) BNC cables (8120-1839).
- (2 off) SMA cables (8120-4948).

AUX. out of Generator 1 on the dual generator module is connected to the clock input of the analyzer and MAIN out of the Generator on the Generator/Analyzer module is connected to Data in 1 of the analyzer. The oscilloscope input which was connected to AUX. out of generator 1is now connected to the inverted main output of the generator on the analyzer/generator module. This allows us to see the (inverted) Detect Enable pulse.

Figure 36 Hardware Test Setup for Getting Started Example 3



1 Connect up the system as shown in Figure 36.

Test Procedure

This procedure continues from the system setting created in example 2. You can either continue directly from example 2 (or load the setting you saved in example 2), or you can load the delivered example 2 setting from the database (HPexamples folder, ‘gsm_example_2’ setting).

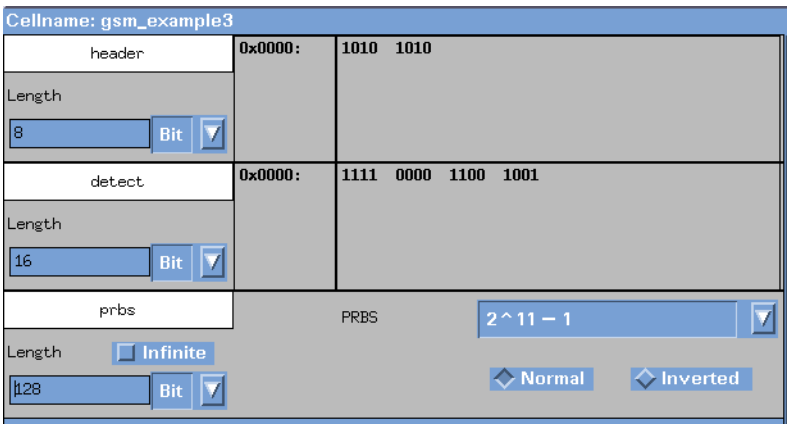
Create cell with detect segment

1 Create a new cell with the following segments:

Table 9 **Segments in cell ‘gsm_example3’**

Type	Name	Length	Contents
Pattern	header	8	1010 1010
Pattern	detect segment	16	111 000 1100 1001
PRBS(pure)	prbs	128	2^{11}-1(normal)

Figure 37 **When complete editor should look as follows:**



2 Save this cell as ‘gsm_example3’.

3 Associate the cell to the generator channel of LT 1.

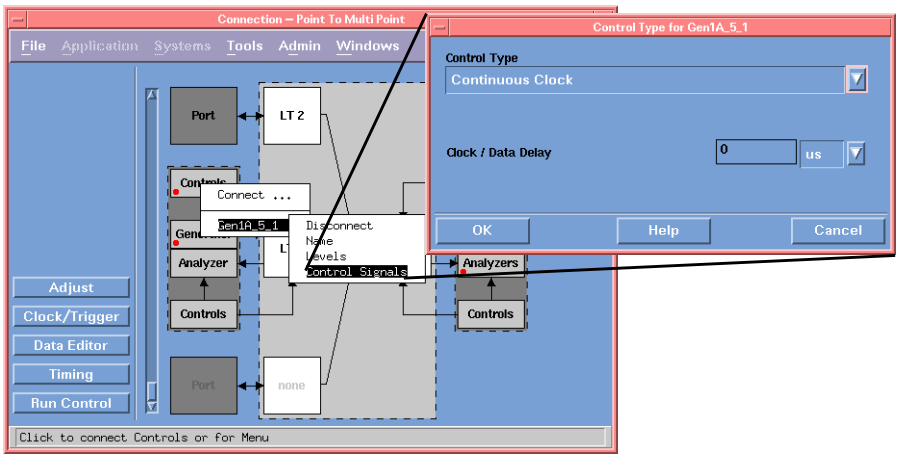
NOTE: For information on how to create a cell and associate it to a generator channel, see “Example 2: Using the Cell Editor to Create Cells” on page 41

Connect Control Channels

The detect enable and external clock signals are generated as control channels in this example.

1 Select a Continuous Clock control signal type for the Generator Controls channel of LT1 using the Control Type window. In this example the auxiliary channel of Generator 2 is physically connected to clock input of the analyzer.

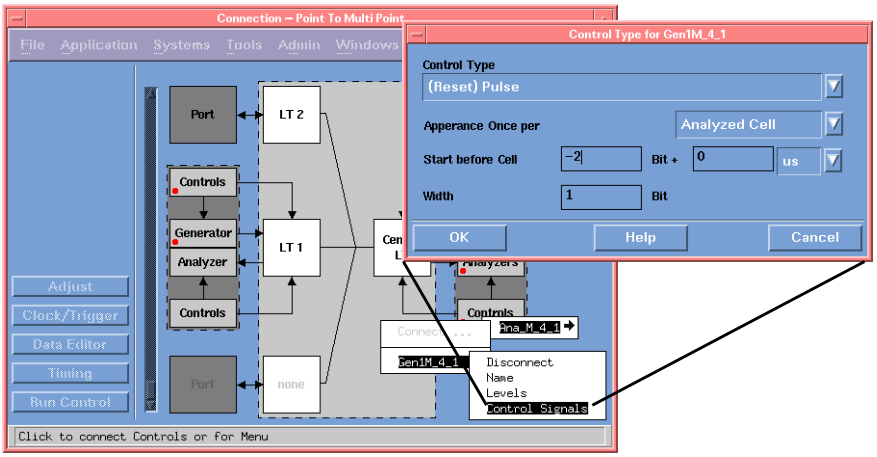
Figure 38 Generating an external clock signal.



2 Connect the Analyzer Controls channel of the Central LT to ‘Gen1M_4_1’. In this example the output channel of the Generator/Analyzer module is physically connected to the detect enable input of the analyzer.

3 Select a (Reset)Pulse control signal type starting -2 bit periods before (2 bit periods after) the cell and a width of 1 bit period. Select Appearance Once per ‘Analyzed Cell’

Figure 39 Generating an external clock signal.



4 Set the voltage level of the analyzer control to 1.0V (Controls>Ana>Gen>level)and enable the non-inverted and inverted outputs. The inverted output is connected to the second input of the scope.

Voltage Levels for the Analyzer Channel

For each analyzer channel you want to use you must set up the input threshold and termination voltages.

1 With the left mouse button, click on the Analyzer that you connected to the Central LT. Keeping the mouse button pressed, select the '<channel name>-> Levels' option and release the mouse button to open up the Voltage Levels window.

2 Select 'Custom' for each Input Level

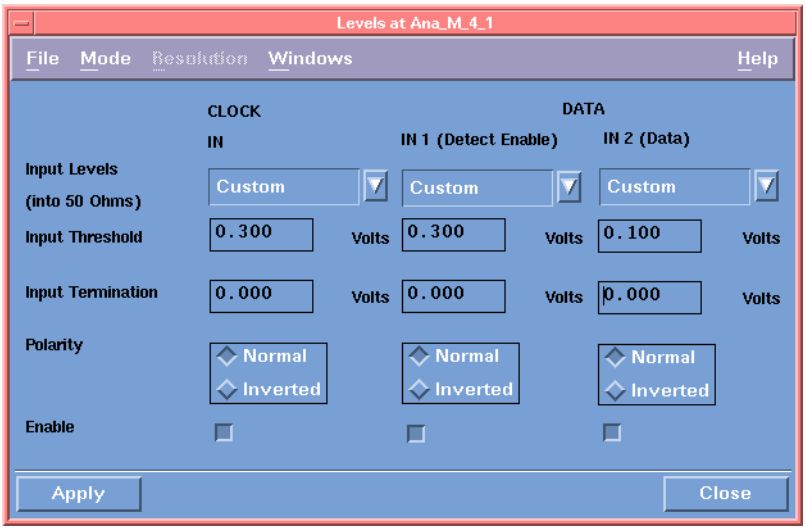
3 The Input Thresholds should be 0.300 volts for CLOCK IN and DATA IN 1(Detect Enable) and 0.10 volts for DATA IN 2.

4 The Input Termination Voltages should be 0.00 volts.

5 Ensure 'Normal Polarity' is selected for all 3 inputs.

6 Enable each input by clicking on the 'Enable' option.

Figure 40 The completed Voltage Levels window should look as follows:



7 Click Apply to download the values to the module

The indicators on the module front panel immediately above each connector should light up immediately after clicking the Apply button.

8 Click Close to return to the Main Connection window.

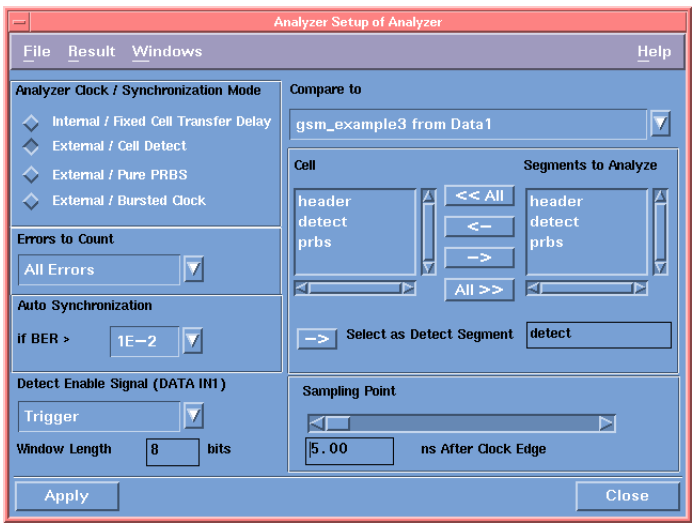
Setting Up the Analyzer

The Analyzer Setup window is used to define:

- synchronization mode of the analyzer
- the type of Detect Enable signal
- which segment is the detect segment
- sampling point of the data with respect to the clock input

1 Open the Analyzer Setup window using the ‘Analyzer->Setup’ option (see Figure 41)

Figure 41 Analyzer Setup Window



2 Select the External / Cell Detect mode.

3 Select the ‘All Errors’ option.

4 Select a BER threshold of 1e-2 for synchronization.

5 Select a Trigger Detect Enable Signal with a window length of 8 bit periods.

Trigger mode means the analyzer starts looking for the detect segment on the rising edge of the detect enable signal and the detect segment must start within the window length.

Gate mode means the start of the detect segment must be within the detect enable signal.

6 Select the ‘gsm_example3 from Gen1M_5_1_Cell’ from the ‘Compare to’ field.

7 Select the ‘detect’ segment in the Cell field, and click the Select as Detect Segment button.

8 The Sampling point may be left at 5 ns.

9 Click Apply to download the values to the module, and then click Close.

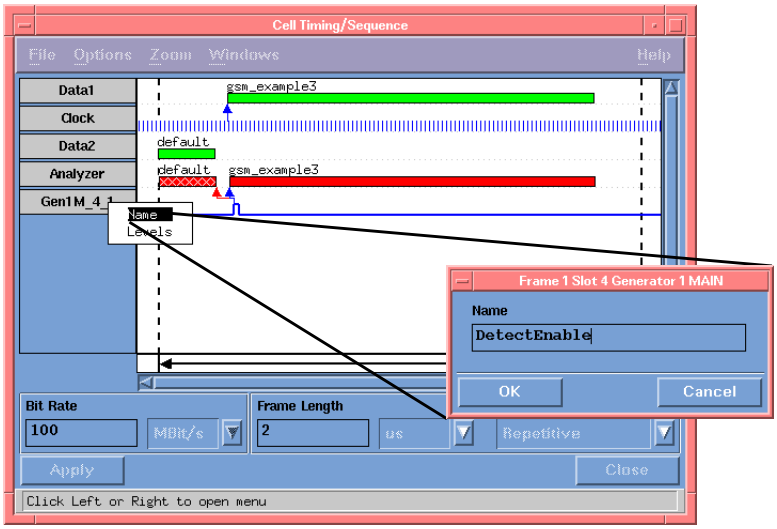
Timing

The cells used in this example will not fit in the 1 µs frame length used in example 2.

1 Change the Frame Length to 2 µs.

2 Change signal names to reflect functionality by clicking on the signal name at the left of the timing window and selection ‘Name’ as shown in Figure 42.

Figure 42 Cell timing for Example 3



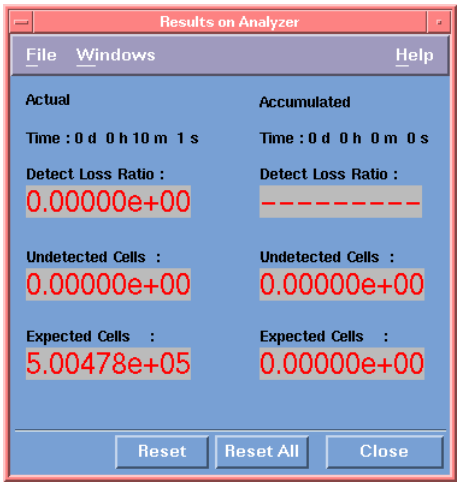
3 Click [Apply] to download the new values to the modules.

Run the test and verify results

Detect Loss Ratio Window

In addition to the BER Results and BER Results Log windows there is a Detect Loss Ratio Window and a Detect Loss Ratio Log window. Typical results are shown in Figure 43 and Figure 44. The Detect Loss Ratio window is opened in the same way as the Results window except Detect Loss Ratio is selected. With everything set up correctly the ‘Actual’ and ‘Accumulated’ detect loss ratios should be 0.

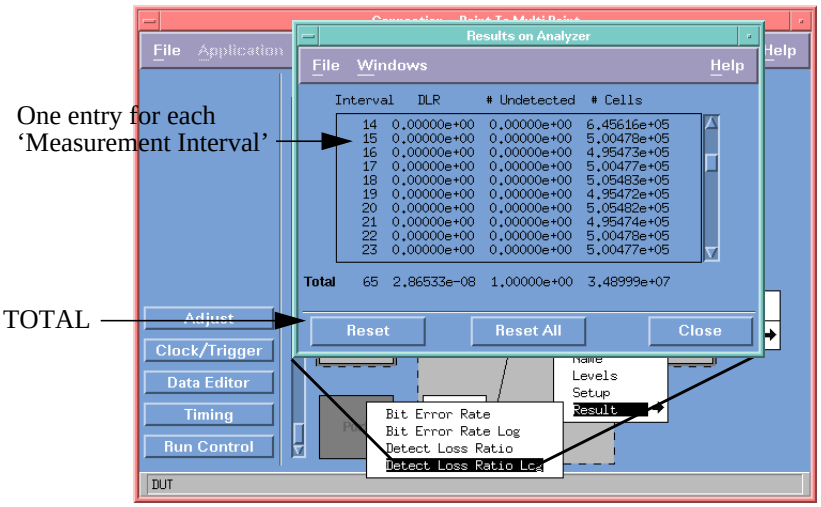
Figure 43 Detect Loss Ratio Window



Detect Loss Ratio Log Window

The Results Log window shown in Figure 44. It is opened in the same way as the Results window except Detect Loss Ratio Log is selected. This displays a complete history of each Measurement Interval since the start of the measurement.

Figure 44 Detect Loss Ratio Log Window



Appendix A

Program Control Functions

Concept

Of general interest

The DVT software programming interface consists of a shared library and a definition file. These can be used by a C, or C++ Compiler, or by HP VEE.

The library is a collection of compiled functions combined into a single file. The definition file defines the data types used by the interface

To use the DVT programming interface from HP VEE, you must import the library and definition file with the Import Library object, and use the Call Function object to call the DVT interface functions.

When writing C or C++ control programs that call DVT library functions, the library file must be specified in the compiler command line using the -lxx option. For example, if a C-program calls an interface function from the DVT - library the compiler must be invoked with following command line options:

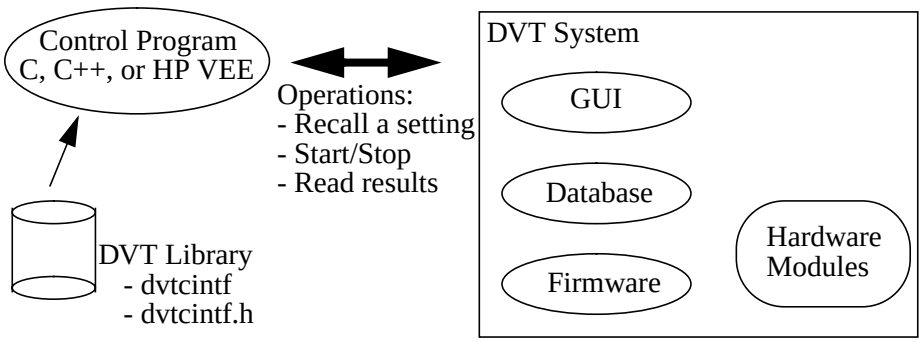
```
cc -Aa program.c -ldvtcintf -ooutputfile
```

The definition file **`dvtcintf.h`** defines the types of data that are passed between the DVT - interface functions and the controlling program. It also defines the type of data the interface functions return, the function names, and the arguments the functions accept.

To use the DVT programming interface you must install the appropriate software tool (e.g. C or C++ compiler, or HP VEE). For a list of available HP software products, please refer to the Installation Guide, Chapter 3, Programming Interface.

DVT - Programming interface

As already mentioned, the DVT-Programming interface is a collection of compiled functions. The DVT library **`dvtcintf`** provides these functions for use in a control program. The definition file **`dvtcintf.h`** defines the data types used.



These two files, are located in following directories on the target system:

`/opt/dvt/lib/libdvtcintf.sl`

`/opt/dvt/include/dvtcintf.h`

The DVT system consists of the following processes:

- graphical user interface (GUI)
- data base server
- firmware server

Currently it is not possible to start the DVT processes using only the programming interface. The control program can only communicate with an up and running initialized DVT system. The DVT system must be started and the application (for example, TDMA) selected, and all parameter and application specific states must be set (or stored within a database setting) using the graphical user interface. After this is done, master control must be given to the controlling program by selecting the <Program> option in the Tools-Controller window.

The DVT interface supports several operations. These operations are called using the **`dvt_Call`** library function:

“RCL” < folder_name, setting_name >
This has the same function as the <Open Setting> option in the File menu, Main Connection window of the graphical user interface.

“INIT:CONT” < 0 | 1 >
This means INITiate CONTinuous and performs the same function as the Stop or the Start button in the Run Control window of the graphical user interface. (0 to stop, 1 to start)

“FETCH?” < Analyzername >
This query returns four values. The values are the number of compared bits, the number of bits in error, the number of expected cells and the number of undetected cells respectively.

“FETCH:CPDEL?” <Generator Name>< Analyzername >
This query returns the Cell Propagation Delay in seconds as a double value.

“MEAS:CPDEL?” <Generator Name>< Analyzername >
This command initiates a measurement and determines the Cell Propagation Delay. It returns the cell propagation delay in seconds as a double value.

“SENS:SYNC” < Analyzername > ONCE
This command initiates a new synchronization and has the same effect as pressing the ReSync button in the BER Results window of the GUI.

All data items such as folder name (text string), setting name (text string), compared bits (double float) and failed bits (double float), are passed between the control program and the DVT system in a data container. There are several interface functions which deal with data containers. These interface functions are used to build up the data container (e.g. **`dvtAppStringToDataCont`**), or to get information from the data container (e.g. **`dvtGetDoubleFromDataCont`**).

A detailed description of all interface functions follow.

Description of interface functions

This section contains a detailed reference of the program control functions.

Each function is defined as follows:

```
long <function name>( <type> <parameter name>,
                      <type> <parameter name>,...);
```

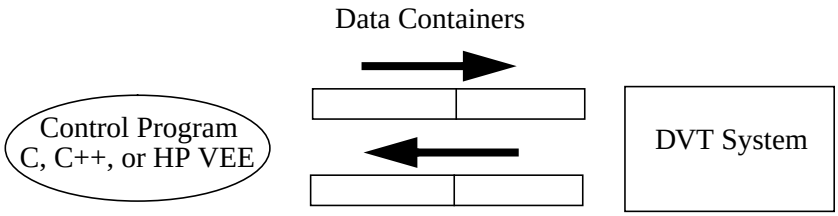
The return value in all functions is an error code (type Long), which is zero if everything is OK. If an error has occurred and an error message is available, then the function returns the value **`ERR_IN_MSG_CONT`** (which is defined as value 100).

Data Types

Data Container
A data container serves as a carrier for a collection of different data types. All system input and output parameters are passed between the control program and the DVT system in a data container (for example: folder name, setting name, compared bits and bits in error).

There are several functions used to manipulate a data container, for example to append a data item or to get a value of a specified item in a data container.

As most functions use parameters for input and output, you need to use at least two data containers.



Error message container
If the returned error code has the value **`ERR_IN_MSG_CONT`** (100), then an error message is available. The DVT-interface puts this error message in an error message container. The function **`dvtGetSizeErrMsgCont`**, gives the number and size of error messages within the container. You must create a buffer of this size, and then put the message into it using **`dvtGetMsgsFromErrMsgCont`**.

When the DVT programming interface is started, the error message container is initialized to a predefined state.

Summary of all program control functions

openDvtInterface	load library and create DVT programming interface
closeDvtInterface	closes the DVT programming interface
dvtGetSizeErrMsgCont	get number of messages, and number of characters in the longest error message contained in the error message container
dvtGetMsgsFromErrMsgCont	get error messages from the error message container
dvtGetHdlDataCont	return a handle of a data container
dvtGetLenDataCont	get number of items in a data container
dvtGetItemTypeDataCont	get type of the selected item in a data container
dvtGetItemStrLenDataCont	get length of a string item in a data container
dvtClearDataCont	reset a data container
dvtAppStringToDataCont	append a string to a data container
dvtAppLongToDataCont	append a long integer to a data container
dvtGetDoubleFromDataCont	get float type contents of the specified item of the data container
dvtGetApplicationHdl	get application handle
dvtCall	function used to communicate with DVT system, using the following operations: “RCL” (recall a setting) “FETCH?” (read results), “INIT:CONT” (start and stop) “SENS:SYNC”(synchronize) “FETCH:CPDEL?”(read cell propagation delay) “MEAS:CPDEL?”(initiates cell propagation delay measurement)

openDvtInterface

Definition:

```
long openDvtInterface(long calledBy);
```

Description:

This function has to be the first function call in a control program. It loads the libraries and creates the DVT programming interface

Return Codes

Value	Alias	Description
0	**DVT_OK**	Everything is OK
1	**DVT_ERR_LIB_NOT_FOUND**	Can't find the library "libdvtintf.sl
2	**DVT_ERR_LIB_CORRUPT**	The library "libdvtintf.sl" was loaded but it seems to be corrupt.
3	**DVT_ERR_INTERFACE**	The library "libdvtintf.sl" was loaded but, the system couldn't create the programming interface. Maybe not enough memory in the workstation.
4	**DVT_ERR_LIB_NOT_LOADED**	An interface function was called but the library was not loaded.
100	**ERR_IN_MSG_CONT**	There is an error message in the error message container

Parameter:

calledBy: This parameter determines the type of program that uses this interface, for example a C-Program, C++ Program or HP-VEE:

DVT_CALLED_BY_VEE (0)
DVT_CALLED_BY_C (1)
DVT_CALLED_BY_CPP (2)

Example:

```
long applHdl;

/* open DVT-Programming Interface */
openDvtInterface(DVT_CALLED_BY_C);

/* Get Application Handle */
dvtGetApplicationHdl(&appHdl, "TDMA", "Mustang", "" );

/* rest of code goes here */
```

closeDvtInterface

Definition:

```
long closeDvtInterface(void);
```

Description:

This is the last function call which has to be made by the user. It cleans up all internal allocated memory, unloads the libraries and closes the DVT programming interface.

Return code:

Value	Alias	Description
0	DVT_OK	Everything is OK
5	DVT_ERR_AT_CLOSE_INTERFACE	An error occurred during clean up of the interface.
100	ERR_IN_MSG_CONT	There is an error message in the error message container

Example:

```
long applHdl, returnCode;

/* open DVT-Programming Interface */
openDvtInterface(DVT_CALLED_BY_C);

/* Get Application Handle */
dvtGetApplicationHdl(&appHdl, "TDMA", "Mustang", "" );

/* rest of code goes here */

returnCode=closeDvtInterface();
```

dvtGetSizeErrMsgCont

Definition:

```
long dvtGetSizeErrMsgCont (  long *nrOfMsgs,
                             long *lenOfLongestMsg);
```

Description:

The number of error messages and the number of characters in the longest error message (including the “End of String \0”) contained in the error message container, are passed back by this function.

This function is called prior to reading the error message container (using **`dvtGetMsgsFromErrMsgCont`**). It provides the size of the buffer required to store the messages.

Return code:

Value	Alias	Description
0	`DVT_OK`	Everything is OK

Parameters:

*nrOfMsgs: number of messages in the error message container

*lenOfLongestMsg: number of characters in the longest error message

Example:

```
long  len, numMsgs;
char  **msgList;
int   i;

dvtGetSizeErrMsgcont(&numMsgs, &len);
if (numMsgs)
   {
   msgList=malloc (sizeof(char *) * numMsgs);
   for (i=0; i<len, i++)
     msgList[i]=malloc (sizeof (char) * len);

   dvtGetMsgsFromErrMsgCont (msgList);
   for (i=0; i<len; i++)
    fprintf(stderr, “Message <%d>: <%s>\n”,i,msgList[i]);
   for (i=0; i<numMsgs; i++)
     free (msgList[i]);
   free (msgList);

   }
```

dvtGetMsgsFromErrMsgCont

Definition:

```
long dvtGetMsgsFromErrMsgCont (char **errorMessages);
```

Description:

This function gets the error messages in the error message container and puts them in the buffers pointed to by **errorMessages.

Return code:

Value	Alias	Description
0	DVT_OK	Everything is OK

Parameter:

**errorMessages: pointer to an array of characters, where the system puts the error messages

Example:

```
long  len, numMsgs;
char  **msgList;
int   i;

dvtGetSizeErrMsgcont(&numMsgs, &len);
if (numMsgs)
   {
   msgList=malloc (sizeof(char *) * numMsgs);
   for (i=0; i<len, i++)
     msgList[i]=malloc (sizeof (char) * len);

   dvtGetMsgsFromErrMsgCont (msgList);
   for (i=0; i<len; i++)
    fprintf(stderr, "Message <%d>: <%s>\n",i,msgList[i]);
   for (i=0; i<numMsgs; i++)
     free (msgList[i]);
   free (msgList);

   }
```

dvtGetHdlDataCont

Definition:

```
long dvtGetHdlDataCont (long *dataContHdl);
```

Description:

This function returns a handle to a “DVT data container”. The container serves as a carrier for a collection of different data types. The data container handle is used by the **`dvtCall`** function to pass input and output parameters to the DVT system. One data container is required for input parameters and a separate container for output parameters.

Some **`dvtCall`** function call operations (i.e. RCL and INIT:CONT) only need one data container. For these, the second handle can be set to zero (See “dvtCall” on page 75.)

Return code:

Value	Alias	Description
0	**`DVT_OK`**	Everything is OK
100	**`ERR_IN_MSG_CONT`**	There is an error message in the error message container

Parameter:

*dataContHdl: Handle of a data container

Example:

```
long dataCont, applHdl;

/* Get Application Handle */
dvtGetApplicationHdl(&appHdl, "TDMA", "Mustang", "" );

/* create data container */
dvtGetHdlDataCont(&dataCont);

/* build up data container with input parameters */
dvtAppStringToDataCont (dataCont, "HPexamples");
dvtAppStringToDataCont (dataCont, "gsm_example_1");

/* Open database setting */
dvtCall (appHdl, "RCL", dataCont, 0, 1);

/* rest of code goes here */
```

dvtGetLenDataCont

Definition:

```
long dvtGetLenDataCont ( long dataContHdl,
                         long *lenCont);
```

Description:

This function passes back the number of items in a data container.

This function (along with **`dvtGetItemTypeDataCont`**) is useful for building more generic functions which use different data types.

Return code:

Value	Alias	Description
0	**`DVT_OK`**	Everything is OK
100	**`ERR_IN_MSG_CONT`**	There is an error message in the error message container

Parameters:

dataContHdl: Handle of a data container

*lenCont: Number of items in the data container

Example:

```
long returnCode;
long dataHdl, numItems;

returnCode=dvtGetHdlDataCont (&dataHdl);

returnCode=dvtGetLenDataCont (dataHdl, &numItems);
```

dvtGetItemTypeDataCont

Definition:

```
long dvtGetItemTypeDataCont (  long dataContHdl,
                               long index,
                               long *type);
```

Description:

This function passes back the type of the selected item.

This function (along with **`dvtGetLenDataCont)`** is useful for building more generic functions which use different data types.

Return code:

Value	Alias	Description
0	`DVT_OK`	Everything is OK
100	`ERR_IN_MSG_CONT`	There is an error message in the error message container

Parameters

dataContHdl: Handle of a data container

index: item in the data container

*type: type of the selected item:

DVT_NULL (0): no entry
DVT_STRING (1): String
DVT_LONG (2): Long
DVT_DOUBLE (3): Double
DVT_INTEGER (4): Integer

NOTE: With the current software revision, only string, long and double data types can be used within data containers.

Example:

```
long returnCode;
long dataHdl, index, dataType;
index=1;

returnCode=dvtGetHdlDataCont (&dataHdl);

returnCode=dvtGetItemTypeDataCont(dataHdl,index,&dataType);
```

dvtGetItemStrLenDataCont

Definition:

```
long dvtGetItemStrLenDataCont (long dataContHdl,
                               long index,
                               long *strLen);
```

Description:

This function returns the string length of the selected string item in the data container.

Return code:

Value	Alias	Description
0	DVT_OK	Everything is OK
100	ERR_IN_MSG_CONT	There is an error message in the error message container

Parameters

dataContHdl: Handle of a data container

index: item in the data container where the string is located

*strlen: length of the string

Example:

```
long dataCont, applHdl, strlen;

/* Get Application Handle */
dvtGetApplicationHdl(&appHdl, "TDMA", "Mustang", "" );

/* create data container */
dvtGetHdlDataCont(&dataCont);

/* build up data container with input parameters */
dvtAppStringToDataCont (dataCont, "HPexamples");
dvtAppStringToDataCont (dataCont, "gsm_example_1");

/* after calling this function, strlen contains 11 */
dvtGetItemStrLenDataCont(dataCont,1,&strLen);

/* after calling this function, strlen contains 14 */
dvtGetItemStrLenDataCont(dataCont,2,&strLen);

/* rest of code goes here */
```

dvtClearDataCont

Definition:

```
long dvtClearDataCont (long dataContHdl);
```

Description:

This function initializes the selected data container. A data container is automatically initialized when it is created. Use this function before re-using a data container.

Return code:

Value	Alias	Description
0	DVT_OK	Everything is OK
100	ERR_IN_MSG_CONT	There is an error message in the error message container

Parameter:

dataContHdl: Handle of the data container to be cleared

Example:

```
long dataCont, applHdl;

/* Get Application Handle */
dvtGetApplicationHdl(&appHdl, "TDMA", "Mustang", "" );

/* Get data container handle */
dvtGetHdlDataCont(&dataCont);

/* build up data container with start value (1) */
dvtAppLongToDataCont (dataCont, 1);
dvtCall (appHdl, "INIT:CONT", dataCont, 0, 1);

/* rest of code goes here */

/* re-initialize data container */
dvtClearDataCont (dataCont);

/* build up data container with stop value (0)*/
dvtAppLongToDataCont (dataCont, 1);
dvtCall (appHdl, "INIT:CONT", dataCont, 0, 1);

/* close DVT program control interface */
closeDvtInterface();
```

dvtAppStringToDataCont

Definition:

```
long dvtAppStringToDataCont (   long dataContHdl,
                                char *parameterValue);
```

Description:

This function appends a string to the specified container.

The append is used to build up a data container prior to calling the dvtCall function. The data container is used to hold the input parameters for the operation (for example: the folder and setting name for the RCL operation).

Return code:

Value	Alias	Description
0	DVT_OK	Everything is OK
100	ERR_IN_MSG_CONT	There is an error message in the error message container

Parameters:

dataContHdl: Handle of the data container which the String is to append to

*parameterValue: Text to append

Example:

```
long dataCont, applHdl;

/* Get Application Handle */
dvtGetApplicationHdl(&appHdl, "TDMA", "Mustang", "" );

/* Get data container handle */
dvtGetHdlDataCont(&dataCont);

/* build up data container with input parameters */
dvtAppStringToDataCont (dataCont, ”HPexamples”);
dvtAppStringToDataCont (dataCont, ”gsm_example_1”);

/* Open database setting */
dvtCall (appHdl, "RCL", dataCont, 0, 1);

/* rest of code goes here */
```

dvtAppLongToDataCont

Definition:

```
long dvtAppLongToDataCont (   long dataContHdl,
                              long parameterValue);
```

Description:

This function appends a long integer to the selected container.

The append is used to build up a data container prior to using the dvtCall function. The data container is used to hold the input parameters for the dvtCall operation (for example: starting or stopping the module).

Return code:

Value	Alias	Description
0	DVT_OK	Everything is OK
100	ERR_IN_MSG_CONT	There is an error message in the error message container

Parameters:

dataContHdl: Handle of the data container which the long is to append to

parameterValue: Long to append

Example:

```
long dataCont, applHdl;

/* Get Application Handle */
dvtGetApplicationHdl(&appHdl, "TDMA", "Mustang", "" );

/* Get data container handle */
dvtGetHdlDataCont(&dataCont);

/* build up data container with start value (1) */
dvtAppLongToDataCont (dataCont, 1);
dvtCall (appHdl, "INIT:CONT", dataCont, 0, 1);

/* rest of code goes here */

/* re-initialize data container */
dvtClearDataCont (dataCont);

/* build up data container with stop value (0)*/
dvtAppLongToDataCont (dataCont, 1);
dvtCall (appHdl, "INIT:CONT", dataCont, 0, 1);

/* close DVT program control interface */
closeDvtInterface();
```

dvtGetDoubleFromDataCont

Definition:

```
long dvtGetDoubleFromDataCont (   long dataContHdl,
                                  long index,
                                  double *parameterValue);
```

Description:

This function enables you to extract floating point type parameters from a data container (for example, the number of compared bits and number of bits in error).

Return code:

Value	Alias	Description
0	DVT_OK	Everything is OK
100	ERR_IN_MSG_CONT	There is an error message in the error message container

Parameters:

dataContHdl: Handle of the data container which contains the double float value

index: The position of the parameter within the data container. The first value in a data container has index 1.

*parameterValue: A pointer to the double float value requested

Example:

```
long dataCont1, dataCont2, applHdl;
double compared_bits, failed_bits;

/* code to open interface & application, create data */
/* containers, recall setting, start module, get refer- */
/* ence bits goes in here */

/* initialize input and output data containers */
dvtClearDataCont (dataCont1);
dvtClearDataCont (dataCont2);

/* build up input data container with parameters */
dvtAppStringToDataCont (dataCont1, "Ana_M_3_1");

dvtCall (appHdl, "FETCH?", dataCont1, dataCont2, 1);

/* get values of compared and failed bits */
dvtGetDoubleFromDataCont(dataCont2, 1, &compared_bits);
dvtGetDoubleFromDataCont(dataCont2, 2, &failed_bits);

/* code to compute BER goes here */
```

dvtGetApplicationHdl

Definition:

```
long dvtGetApplicationHdl (  long *appHandle,
                             char *applicationName,
                             char *systemname,
                             char *optionalFeatures);
```

Description:
This function gets a handle to communicate with an application. This handle is used to identify the application within other function calls.

Return code:

Value	Alias	Description
0	**DVT_OK**	Everything is OK
100	**ERR_IN_MSG_CONT**	There is an error message in the error message container

Parameters:

applicationName: Currently must be “TDMA”

systemName: the name of the system (this is equivalent to selecting a system from the DVT Main Connection window; the systems are configured in the dvt.sys file)

optionalFeatures: for future use (use any string to initialize)

appHandle: A handle which identifies the application.

Example:

```
long applHdl;

/* open DVT-Programming Interface */
openDvtInterface(DVT_CALLED_BY_C);

/* Get Application Handle */
dvtGetApplicationHdl(&appHdl, "TDMA", "Mustang", "" );

/* rest of code goes here */
```

dvtCall

Definition:

```
long dvtCall ( long appHandle,
               char *operation,
               long inputDataContHdl,
               long outputDataContHdl,
               long transaction);
```

Description:
This function enables the control program to communicate with the DVT system. For example to recall a system setting, to start or stop the module, and to retrieve number of compared and bits in error.

All input and output parameters such as folder name, setting name, compared bits and failed bits, are passed between the system hardware and the user program in a data container. Use the interface functions to build up the data container (e.g. **`dvtAppStringToDataCont`**), or to retrieve data from the data container (e.g. **`dvtGetDoubleFromDataCont`**).

Return code:
Error code: 100 if there is an error message, which is in the error message container.

Parameter:

appHandle:	Handle to talk to the chosen application (see function above)
*operation:	The following operations are possible: **“RCL”**, used to recall a system setting from the database **“INIT:CONT**”, used to start/stop a module **“FETCH?”**, used to get values from the module **“FETCH:CPDEL?”**, used to get the propagation delay **“MEAS:CPDEL?”**, used to measure the cell propagation delay **“SENS:SYNC”**, used to synchronize
inputDataContHdl:	Handle for the input parameter data container
outputDataContHdl:	Handle for the output parameter data container
transaction:	For future use (currently set to 1)

“RCL”
This operation opens an existing database setting. It has the same function as the <Open Setting> File menu option, in the Main Connection window. The input data container must contain the name of the folder and the name of the setting (first entry: folder name, second entry: setting name).

The output data container is not used.

Example:

```
long dataCont, applHdl;

/* Get Application Handle */
dvtGetApplicationHdl(&appHdl, "TDMA", "Mustang", "" );

/* Get data container handle */
dvtGetHdlDataCont(&dataCont);

/* build up data container with input parameters */
dvtAppStringToDataCont (dataCont, ”HPexamples”);
dvtAppStringToDataCont (dataCont, ”gsm_example_1”);

/* Open database setting */
dvtCall (appHdl, "RCL", dataCont, 0, 1);

/* rest of code goes here */
```

“INIT:CONT”

This operation starts or stops the modules. It has the same function as the Start or Stop button in the run control window of the graphical user interface.

If you want to start the modules create a data container with the value 1 (type long). To stop the modules use a data container with value 0 (type long)

Example:

```
long dataCont, applHdl;

/* Get Application Handle */
dvtGetApplicationHdl(&appHdl, "TDMA", "Mustang", "" );

/* Get data container handle */
dvtGetHdlDataCont(&dataCont);

/* build up data container with start value (1) */
dvtAppLongToDataCont (dataCont, 1);
dvtCall (appHdl, "INIT:CONT", dataCont, 0, 1);

/* rest of code goes here */

/* re-initialize data container */
dvtClearDataCont (dataCont);

/* build up data container with stop value (0)*/
dvtAppLongToDataCont (dataCont, 1);
dvtCall (appHdl, "INIT:CONT", dataCont, 0, 1);

/* close DVT program control interface */
closeDvtInterface();
```

“FETCH?”

This query returns the number of compared bits, failed bits, expected cells and undetected cells in the output data container.

The input data container must contain the analyzer name.

To calculate the bit error rate you must first retrieve the reference number of compared and failed bits. Then start the measurement. Then retrieve the number of compared and failed bits a second time, calculate the difference between these and the reference and then divide the results to get the bit error rate. This is necessary because it is not possible to reset these values. The Detect Loss Ratio is calculated similarly.

Example

```
long dataCont1, dataCont2, applHdl;
double compared_bits, failed_bits, expected_cells,
     undetected_cells;
double abs_compared_bits, abs_failed_bits,
     abs_expected_cells, abs_undetected_cells;
double ref_compared_bits, ref_failed_bits,
     ref_expected_cells, ref_undetected_cells;
int i, no_seconds=3600;

/* code to open interface & application, create data */
/* containers, recall setting, start module, goes in here */

/* put analyzer name in data container */
dvtAppStringToDataCont (dataCont1, "Ana_M_3_1");

/* get reference values from DVT system*/
dvtCall (appHdl, "FETCH?", dataCont1, dataCont2, 1);
dvtGetDoubleFromDataCont(dataCont2, 1, &ref_compared_bits);
dvtGetDoubleFromDataCont(dataCont2, 2, &ref_failed_bits);
dvtGetDoubleFromDataCont(dataCont2, 2, &ref_expected_cells);
dvtGetDoubleFromDataCont(dataCont2, 2,
     &ref_undetected_cells);

for (i=0; i<no_seconds; ++i) {
     sleep (1);
     /* get values of compared and failed bits and
          expected and undetected cells*/
     dvtCall (appHdl, "FETCH?", dataCont1, dataCont2, 1);

     /* get values of compared and failed bits */
     dvtGetDoubleFromDataCont(dataCont2, 1, &compared_bits);
     dvtGetDoubleFromDataCont(dataCont2, 2, &failed_bits);
     dvtGetDoubleFromDataCont(dataCont2, 2,
          &expected_cells);
     dvtGetDoubleFromDataCont(dataCont2, 2,
          &undetected_cells);
```

```
        /* compute bit error rate and detect loss ratio */
        abs_compared_bits = compared_bits - ref_compared_bits;
        abs_failed_bits = failed_bits - ref_failed_bits;
         abs_expected_cells = expected_cells
              - ref_expected_cells);
         abs_undetected_cells = undetected_cells
              - ref_undetected_cells);
         printf("Bit Error Rate=%f\n\n",abs_failed_bits/
              abs_compared_bits);
         printf("Detect Loss Ratio=%f\n\n",
              abs_undetected_cells/abs_expected_cells);
}
```

"FETCH:CPDEL?"

This query returns the cell propagation delay between the specified generator and analyzer in the output data container. This query does not initiate a measurement, this is done by "MEAS:CPDEL?"

The input data container must contain the generator and analyzer names.

Example

```
long dataCont1, dataCont2, applHdl;
double cell_propagation_delay;

/* code to open interface & application, create data */
/* containers, recall setting, start module, goes in here */

/* put generator and analyzer names in data container */
dvtAppStringToDataCont (dataCont1, "Gen_M_4_1");
dvtAppStringToDataCont (dataCont1, "Ana_M_3_1");

/* get reference values from DVT system*/
dvtCall (appHdl, "FETCH:CPDEL?", dataCont1, dataCont2, 1);
dvtGetDoubleFromDataCont(dataCont2, 1,
     &cell_propagation_delay);

printf("Cell Propagation Delay=%f seconds\n\n",
     cell_propagation_delay);
```

"MEAS:CPDEL?"

This command initiates a measurement of the cell_propagation_delay and returns the value in the output data container.

The input data container must contain the generator and analyzer names.

Example

```
long dataCont1, dataCont2, applHdl;
double cell_propagation_delay;
```

```
/* code to open interface & application, create data */
/* containers, recall setting, start module, goes in here */

/* put generator and analyzer names in data container */
dvtAppStringToDataCont (dataCont1, "Gen_M_4_1");
dvtAppStringToDataCont (dataCont1, "Ana_M_3_1");

/* get reference values from DVT system*/
dvtCall (appHdl, "MEAS:CPDEL?", dataCont1, dataCont2, 1);
dvtGetDoubleFromDataCont(dataCont2, 1,
     &cell_propagation_delay);

printf("Cell Propagation Delay=%f seconds\n\n",
     cell_propagation_delay);
```

"SENS:SYNC"

This command initiates a new synchronization .

The input data container must contain the analyzer name.

Example

```
long dataCont1, dataCont2, applHdl;
double compared_bits, failed_bits;
double abs_compared_bits, abs_failed_bits;
double prev_compared_bits, prev_failed_bits;
double BER;
int i, no_seconds=3600;

/* code to open interface & application, create data */
/* containers, recall setting, start module, goes in here */

/* put analyzer name in data container */
dvtAppStringToDataCont(dataCont1, "Ana_M_3_1");

/* get reference values from DVT system*/
dvtCall (appHdl, "FETCH?", dataCont1, dataCont2, 1);
dvtGetDoubleFromDataCont(dataCont2, 1, &prev_compared_bits);
dvtGetDoubleFromDataCont(dataCont2, 2, &prev_failed_bits);

for (i=0; i<no_seconds; ++i)
{
     sleep (1);
     /* get values of compared and failed bits */
     dvtCall (appHdl, "FETCH?", dataCont1, dataCont2, 1);
     /* get values of compared and failed bits */
     dvtGetDoubleFromDataCont(dataCont2, 1, &compared_bits);
     dvtGetDoubleFromDataCont(dataCont2, 2, &failed_bits);
     /* compute bit error rate*/
```

```
        abs_compared_bits = compared_bits - prev_compared_bits;
        abs_failed_bits = failed_bits - prev_failed_bits;
         prev_compared_bits = compared_bits;
         prev_failed_bits = failed_bits;
         BER = abs_failed_bits/abs_compared_bits;
         if(BER > 1e-2)
         {
              dvtCall (appHdl, "SENS:SYNC", dataCont1, 0, 1);
              printf("Bit Error Rate=%f! Resynchronizing\n\n",
                   BER);
         }
}
```

C-Program Example

This program opens the DVT-Interface, starts the measurement, takes a reference for the failed and compared bits. It then waits for a pre-defined time to let the measurement run, then gets the number of failed and compared bits and calculates the difference between them and the reference. After this, it calculates the bit error rate by dividing the absolute number of failed bits by the absolute number of compared bits and prints the bit error rate. Finally it stops the measurement and closes the interface.

Remember that you have to start the DVT system GUI and select the Program control before starting the control program. You have to select the application and system that is used in the control program (in this example: application=”TDMA”, system=”Mustang”).

This example can be found with a makefile under the path:
`/opt/dvt/share/examples/prgctrl/c-progs`.

This example should be copied to another directory before modifications are made.

```
/* C Program Example */
#include <stdio.h>
#include <stdlib.h>
#include “/opt/dvt/include/dvtcintf.h”


/* function displayErrMsgs:
function gets and displays error messages if possible */
void displayErrMsgs (long errorcode)
{
   long i;
   long ret;
   long nrMsgs, longestMsg;
   char **msgList;
   if (errorcode==DVT_OK) return;
   if (errorcode!=ERR_IN_MSG_CONT)
   {
     printf(“Error while executing DVT-Interface”
       ” function, no error message \n”);
            return;
   }
  ret=dvtGetSizeErrMsgCont(&nrMsgs, &longestMsg);
  if (ret!=0)
   {
     printf(“Error while executing DVT-Interface function,”
       “no error message \n”);
             return;
   }
  if (nrMsgs > 0)
   {
     msgList=malloc (sizeof(char *) * nrMsgs);
```

```
    for(i=0; i<nrMsgs; i=i+1)
      msgList[i]=malloc (sizeof (char) * longestMsg);
  ret=dvtGetMsgsFromErrMsgCont (msgList);
  if (ret!=0)
    printf("Error while executing DVT-Interface function,"
      "no error message\n");
  else
  {
    for(i=0; i<nrMsgs; i++)
    {
      printf("Errormessage[%d]: <%s>\n",i,msgList[i]);
      free(msgList[i]);
    }
    free(msgList);}
    return;
  }
/* End of function displayErrMsgs  */


main()
{

 /* Variable Declaration */
  long ret;
  int i, no_seconds=10;
  long dataHandle1,dataHandle2;
  long appHdl;
  double compared_bits, failed_bits, expected_cells,
    undetected_cells;
  double ref_compared_bits, ref_failed_bits,
    ref_expected_cells, ref_undetected_cells;
  double abs_compared_bits, abs_failed_bits,
    abs_expected_cells, abs_undetected_cells;
  double prev_compared_bits, prev_failed_bits;
  double bit_error_rate, prev_BER, detect_loss_ratio;
  double cell_propagation_delay;
  char operation_recall[]="RCL";
  char operation_start_stop[]="INIT:CONT";
  char operation_results[]="FETCH?";
  char operation_autoadjust[]="MEAS:CPDEL?";
  char operation_get_delay[]="FETCH:CPDEL?";
  char operation_synchronize[]="SENS:SYNC";
  char analyzername[]="Ana_M_4_1";
  char generatorname[]  ="Gen_M_3_1";
 /* End of Variable Declaration */


 /* open DVT-Interface, exit if error appeared */
  ret= openDvtInterface(DVT_CALLED_BY_C);
  if (ret!=0) {displayErrMsgs(ret);exit(0);}
```

```
 /* End of open DVT-Interface */


 /* Get Application Handle, selected: Application=”TDMA”,
System name=”Mustang”  */
   ret= dvtGetApplicationHdl( &appHdl, “TDMA”, “Mustang”,
     ““);
   if (ret!=0)
   {
     displayErrMsgs(ret);ret=closeDvtInterface();exit(0);
   }
 /* End of Application Handle */



 /* Get two data container handles  */
   ret=dvtGetHdlDataCont(&dataHandle1);
   if (ret!=0)
     {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   ret=dvtGetHdlDataCont(&dataHandle2);
   if (ret!=0)
     {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}
 /* End of Get two data container handles


 /* Open database setting */

   dvtAppStringToDataCont (dataHandle1, “HPexamples”);
   if (ret!=0)
     {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   dvtAppStringToDataCont (dataHandle1, “gsm_example_2”);
   if (ret!=0)
     {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   dvtCall (appHdl, operation_recall, dataHandle1, 0, 1);
   if (ret!=0)
     {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}
/* End Open database setting */


 /* perform auto-adjust*/
   ret = dvtClearDataCont (dataHandle1);
   if (ret!=0)
     {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   ret = dvtAppStringToDataCont (dataHandle1,
     generatorname);
   if (ret!=0)
```

```
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   ret = dvtAppStringToDataCont (dataHandle1, analyzername);
   if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   ret = dvtCall (appHdl, operation_autoadjust, dataHandle1,
      dataHandle2, 1);
   if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   ret = dvtGetDoubleFromDataCont(dataHandle2, 1,
      &cell_propagation_delay);
   if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   printf("\nCell Propagation Delay=%f\n",
      cell_propagation_delay);

 /* end of auto-adjust*/


/* start modules */
   ret = dvtClearDataCont (dataHandle1);
   if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   ret=dvtAppLongToDataCont (dataHandle1, 1);
   if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   ret=dvtCall (appHdl, operation_start_stop, dataHandle1,
      0, 1);
   if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}
 /* End of start modules */


/* get compared and failed bits and store the values as a
reference */
   ret=dvtClearDataCont (dataHandle1);
   if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   ret=dvtAppStringToDataCont (dataHandle1, analyzername);
   if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

   ret=dvtCall (appHdl, operation_results, dataHandle1,
      dataHandle2, 1);
```

```
  if (ret!=0)
    {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

  ret=dvtGetDoubleFromDataCont(dataHandle2, 1,
    &ref_compared_bits);
  if (ret!=0)
    {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

  ret=dvtGetDoubleFromDataCont(dataHandle2, 2,
    &ref_failed_bits);
  if (ret!=0)
    {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

  ret=dvtGetDoubleFromDataCont(dataHandle2, 1,
    &ref_expected_cells);
  if (ret!=0)
    {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

  ret=dvtGetDoubleFromDataCont(dataHandle2, 2,
    &ref_undetected_cells);
  if (ret!=0)
    {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}
  prev_compared_bits = ref_compared_bits;
  prev_failed_bits = ref_failed_bits;
 /* End of get reference  */


 /* get compared and failed bits and display difference
between these values and the references */
  for (i=0; i<no_seconds; ++i)
  {
    sleep(1);

    ret=dvtClearDataCont (dataHandle2);
    if (ret!=0)
    {
      displayErrMsgs(ret);
      ret=closeDvtInterface();
      exit(0);
    }

    ret=dvtCall (appHdl, operation_results, dataHandle1,
      dataHandle2, 1);
    if (ret!=0)
    {
      displayErrMsgs(ret);
      ret=closeDvtInterface();
      exit(0);
    }
```

```
ret=dvtGetDoubleFromDataCont(dataHandle2, 1,
&compared_bits);
if (ret!=0)
{
  displayErrMsgs(ret);
  ret=closeDvtInterface();
  exit(0);
}

ret=dvtGetDoubleFromDataCont(dataHandle2, 2,
  &failed_bits);
if (ret!=0)
{
  displayErrMsgs(ret);
  ret=closeDvtInterface();
  exit(0);
}

ret=dvtGetDoubleFromDataCont(dataHandle2, 1,
&expected_cells);
if (ret!=0)
{
  displayErrMsgs(ret);
  ret=closeDvtInterface();
  exit(0);
}

ret=dvtGetDoubleFromDataCont(dataHandle2, 2,
  &undetected_cells);
if (ret!=0)
{
  displayErrMsgs(ret);
  ret=closeDvtInterface();
  exit(0);
}

abs_compared_bits=compared_bits-ref_compared_bits;
abs_failed_bits=failed_bits-ref_failed_bits;
bit_error_rate=abs_failed_bits/abs_compared_bits;

printf("\nNumber of Failed Bits=%f\n",abs_failed_bits);
printf("Number of Compared Bits=%f\n",
  abs_compared_bits);
printf("Bit Error Rate=%f\n\n",bit_error_rate);

abs_expected_cells=expected_cells-ref_expected_cells;
abs_undetected_cells=undetected_cells-
  ref_undetected_cells;
detect_loss_ratio=abs_undetected_cells/
  abs_expected_cells;
```

```
      printf("\nNumber of Undetected Cells=%f\n",
        abs_undetected_cells);
      printf("Number of Expected Cells=%f\n",
        abs_expected_cells);
      printf("Detect Loss Ratio=%f\n\n",detect_loss_ratio);

      prev_BER = (failed_bits - prev_failed_bits)/
        (compared_bits - prev_compared_bits);
      prev_compared_bits = compared_bits;
      prev_failed_bits = failed_bits;
      if(prev_BER > 1.0e-2)

      ret=dvtCall (appHdl, operation_synchronize,
        dataHandle1, 0, 1);
      if (ret!=0)
      {
        displayErrMsgs(ret);
        ret=closeDvtInterface();
        exit(0);
      }
  /* End of display bit error rate */
    }


  /* stop modules */
    ret=dvtClearDataCont (dataHandle1);
    if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

    ret=dvtAppLongToDataCont (dataHandle1, 0);
    if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}

    ret=dvtCall (appHdl, operation_start_stop, dataHandle1,
      0, 1);
    if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}
  /* End of stop modules */



  /* close DVT-Interface */
    ret= closeDvtInterface();
    if (ret!=0)
      {displayErrMsgs(ret);ret=closeDvtInterface();exit(0);}
  /* End of close DVT   */
}
```

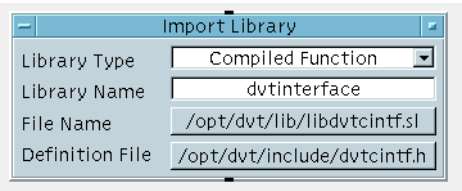
Using the program control functions with HP VEE

Creating and Deleting the DVT-Interface

The HP VEE supports three kinds of user-defined functions, the *UserFunction*, *Compiled Function* and *Remote Function*. All of these functions can be called using the *Call Function* object located in the HP VEE control environment. To use the DVT-library you must import this library with the *Import Library* object, this attaches the shared library to the HP VEE process and parses the definition file declarations. The DVT-interface functions can then be called with the *Call Function* object by specifying the function name (see also HP VEE Advanced Programming Techniques, Chapter 5-17)

Currently it is not possible to start the DVT processes using only the programming interface. The control program can only communicate with an up and running initialized DVT system. The DVT system must be started and the application (for example, TDMA) selected, and all parameter and application specific states must be set (or stored within a database setting) using the graphical user interface. After this is done, master control must be given to the controlling program by selecting the <Program> option in the Tools-Controller window. When the controlling program is finished the <GUI> option must be selected.

Figure 45 Import Library



HP VEE Program Example

This example program is made up of three branches.

The first branch loads the DVT-Interface library and opens the interface.

The second branch gets an application handle and creates two data containers. It then starts the modules, and gets the reference compared and failed bit values. It then continually gets the compared and failed bits and calculates the BER. It stops when the user presses the “EXIT” button. The panel displays the absolute values, which is the difference between the reference and the current values.

The last branch closes the DVT-Interface and unloads the library.

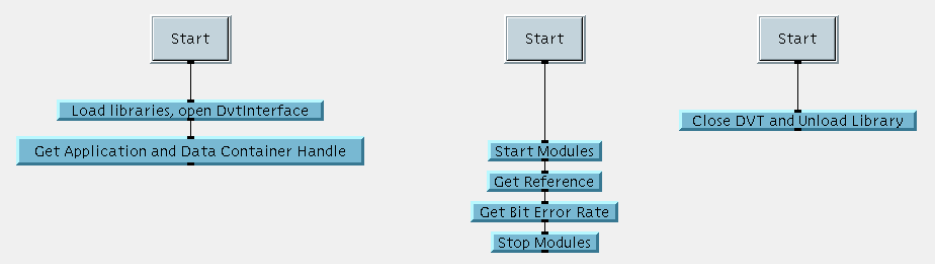
The program is split into three branches because it is only necessary to load and close the library once.

For more details refer to the description of the DVT-Interface functions. This VEE program can be found under the path:

`/opt/dvt/share/examples/prgctrl/VEE-progs/vee_example`

This example should be copied to another directory before any modifications are made.

Figure 46 Main Structure



To Run the Example:

1. Start DVT system and run the TDMA application.
2. Open setting gsm_example_1 from folder HPexamples.
3. Open the Controller window from the Tools..>Controller menu item and select the <Program> option.
4. Open an HP-UX terminal window.
5. Start HP VEE (by typing ‘veetest’).
6. Open File **`vee_example`**.
7. Run the first branch by clicking on “Start” and wait for it to complete.
8. Run the second branch to start measurements. A panel is displayed which shows absolute BER values and contains an “EXIT” button. This button stops the measuremens. This branch may be run as often as is desired.
9. Run the last branch to unload the library.

Figure 47 Load libraries, open DvtInterface

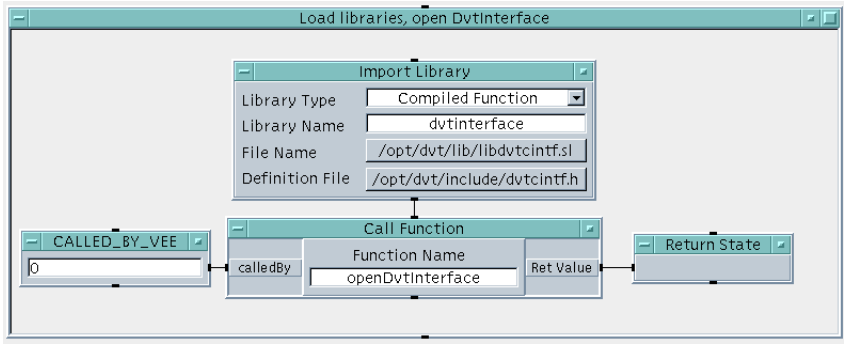


Figure 48 Get Application and Data Container Handle

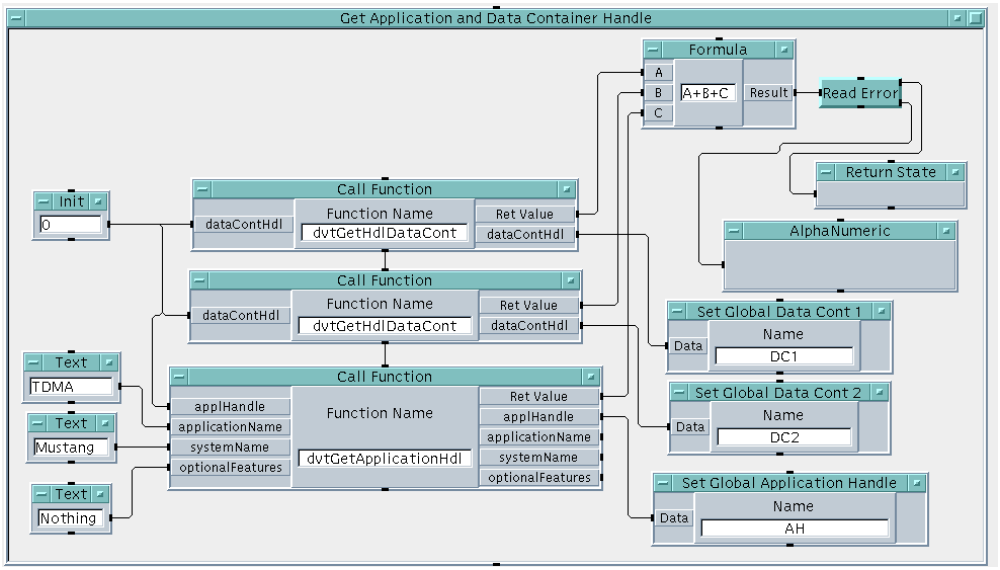


Figure 49 Start Modules

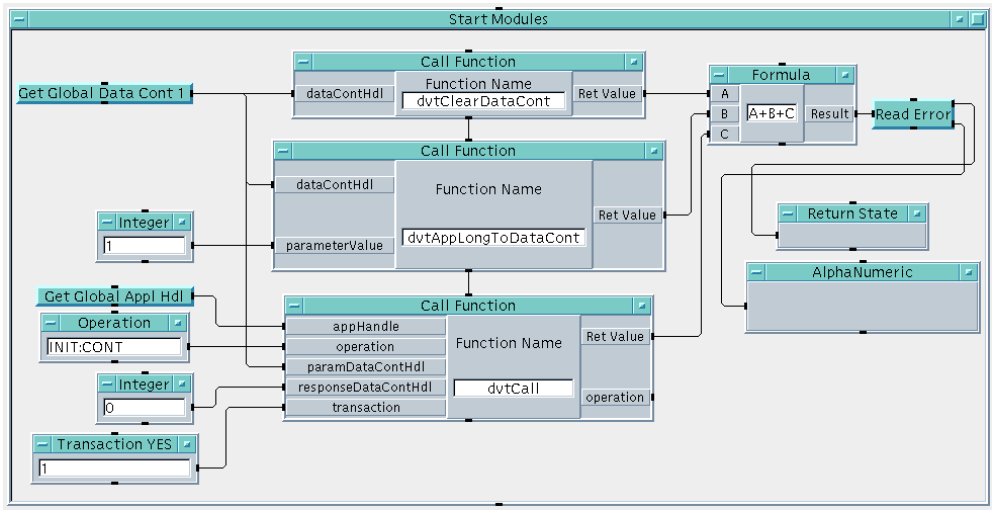


Figure 50 Get Reference

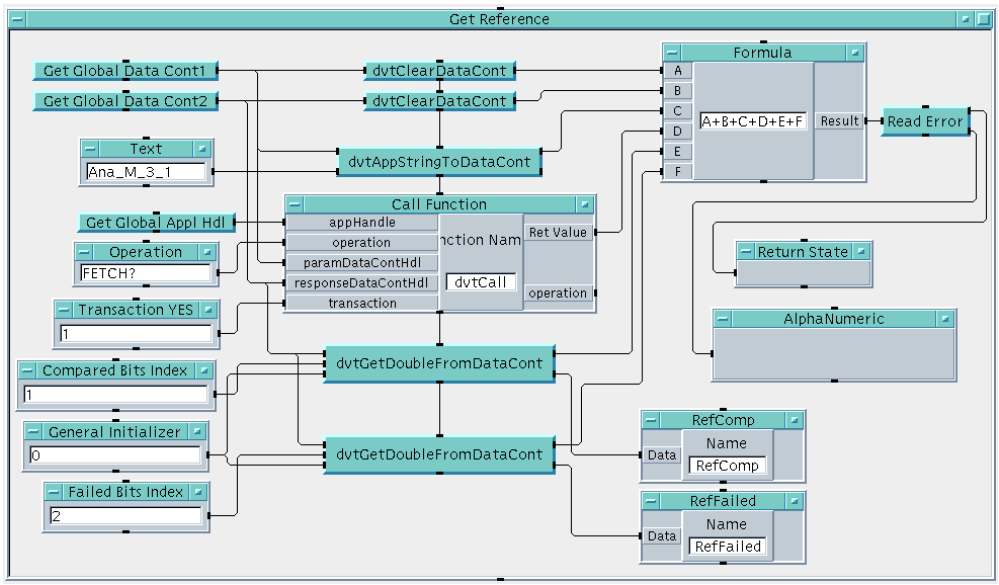


Figure 51 Get Bit Error Rate Detail

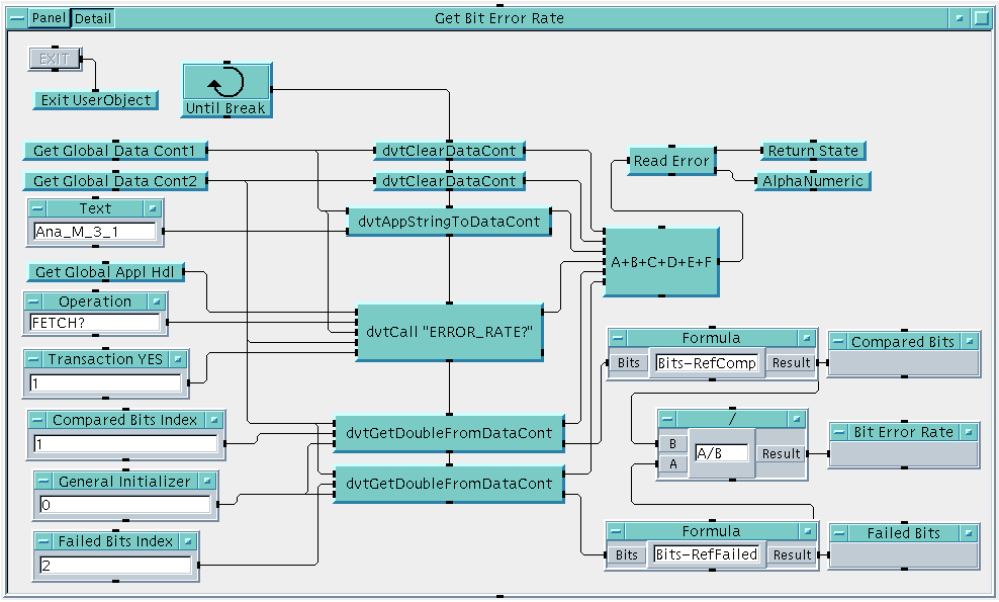


Figure 52 Get Bit Error Rate Panel

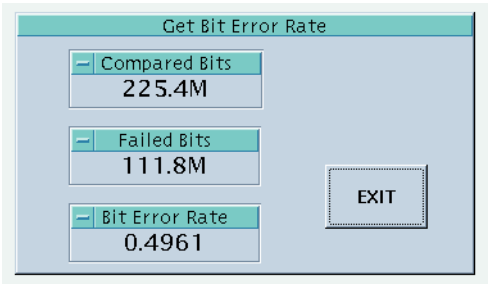


Figure 53 Stop Modules

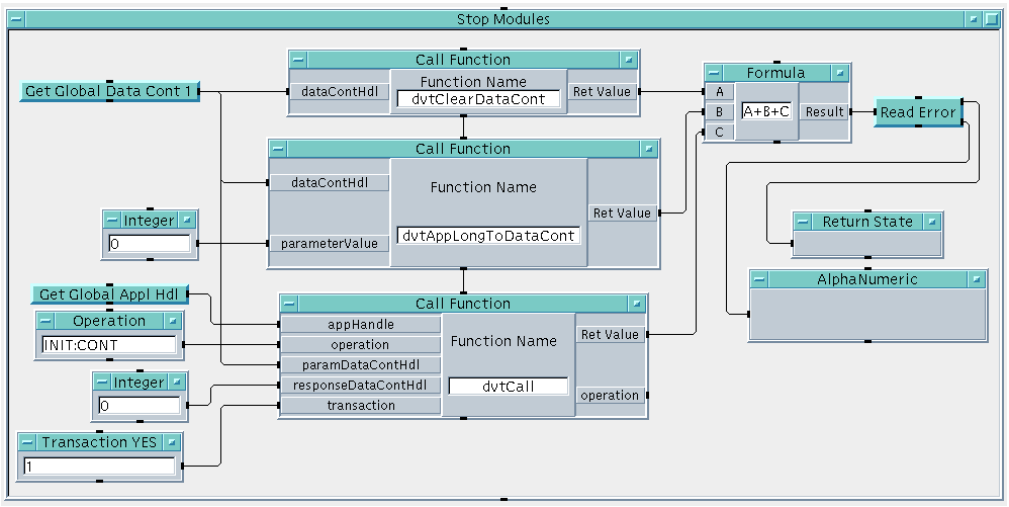
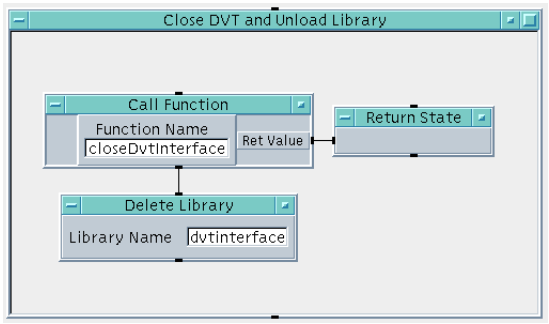


Figure 54 Close DVT and Unload Library



Index

Index

Index