Series 2040 Test Systems

ICAM
User Manual

Part Number 4200-0165
Version 3.0
# Table Of Contents

- System Description ........................................................................................................... 7
  - System Description ........................................................................................................ 9
  - Hardware Overview ....................................................................................................... 9
  - Software Overview ....................................................................................................... 9
  - AutoGen Environment .................................................................................................. 10
    - Parts ......................................................................................................................... 10
    - Nets ......................................................................................................................... 11
    - Basic ......................................................................................................................... 13
  - CYX EXECUTIVE API .................................................................................................. 13
    - Changeable code ....................................................................................................... 14
    - Unchangeable code ................................................................................................... 14
    - AutoGen generated code ......................................................................................... 14
    - Cyclops generated code ......................................................................................... 14
  - Power Supplies ........................................................................................................... 14
  - Vacuum Control System ............................................................................................. 14
- Installation ...................................................................................................................... 18
  - Software ...................................................................................................................... 21
    - Software ................................................................................................................... 22
    - AutoGen ................................................................................................................... 22
      - File Menu/Toolbar ................................................................................................. 23
      - Edit Menu ............................................................................................................... 24
      - Vacuum .................................................................................................................. 25
      - Execute Menu ........................................................................................................ 25
      - Help Menu ............................................................................................................. 29
      - ToolBars ................................................................................................................ 29
      - Matrix Grid ............................................................................................................ 29
      - Matrix Grid Format Changes .............................................................................. 30
  - Parts ............................................................................................................................... 31
    - Part Editor ................................................................................................................ 31
      - Graph ...................................................................................................................... 33
      - Guard Editor .......................................................................................................... 34
        - Parameters .......................................................................................................... 34
  - Nets ............................................................................................................................... 36
    - Net Editor ................................................................................................................ 36
    - Part Pin Editor ......................................................................................................... 38
    - Part O in List ............................................................................................................ 38
  - Shorts Test .................................................................................................................... 39
    - Shorts Test Parameters .......................................................................................... 39
  - Verify Test Fixture ....................................................................................................... 40
  - Assign Channels ......................................................................................................... 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Panel Parameters</td>
<td>41</td>
</tr>
<tr>
<td>Basic</td>
<td>42</td>
</tr>
<tr>
<td>CYX executive API</td>
<td>44</td>
</tr>
<tr>
<td>Changeable Code</td>
<td>44</td>
</tr>
<tr>
<td>Provided Unchangeable Code</td>
<td>45</td>
</tr>
<tr>
<td>Interface to CYX executive</td>
<td>53</td>
</tr>
<tr>
<td>CYX Executive API Initialization</td>
<td>53</td>
</tr>
<tr>
<td>CYX Executive API Execution</td>
<td>53</td>
</tr>
<tr>
<td>CYX Executive API Data Logging</td>
<td>55</td>
</tr>
<tr>
<td>fixture.bas</td>
<td>56</td>
</tr>
<tr>
<td>Vbib32.bas</td>
<td>56</td>
</tr>
<tr>
<td>Niglobal32.bas</td>
<td>56</td>
</tr>
<tr>
<td>Icam32.bas</td>
<td>56</td>
</tr>
<tr>
<td>Directory Structure</td>
<td>56</td>
</tr>
<tr>
<td>Functional Calls</td>
<td>59</td>
</tr>
<tr>
<td>Sample Functional Call</td>
<td>60</td>
</tr>
<tr>
<td>AltSrcRM eas</td>
<td>61</td>
</tr>
<tr>
<td>BetaMeas</td>
<td>63</td>
</tr>
<tr>
<td>CDischarge</td>
<td>65</td>
</tr>
<tr>
<td>Cmeas</td>
<td>67</td>
</tr>
<tr>
<td>CofRCM eas</td>
<td>69</td>
</tr>
<tr>
<td>CofRCM eas2</td>
<td>71</td>
</tr>
<tr>
<td>CofRCM eas3</td>
<td>73</td>
</tr>
<tr>
<td>CPol</td>
<td>75</td>
</tr>
<tr>
<td>DVM eas</td>
<td>77</td>
</tr>
<tr>
<td>D ZM eas</td>
<td>79</td>
</tr>
<tr>
<td>FixID</td>
<td>81</td>
</tr>
<tr>
<td>FixStat</td>
<td>82</td>
</tr>
<tr>
<td>Guard</td>
<td>83</td>
</tr>
<tr>
<td>GuardBd</td>
<td>84</td>
</tr>
<tr>
<td>HFM ux</td>
<td>85</td>
</tr>
<tr>
<td>ICAM Clear</td>
<td>87</td>
</tr>
<tr>
<td>ICAM Delay</td>
<td>88</td>
</tr>
<tr>
<td>ICAM GetSamples</td>
<td>89</td>
</tr>
<tr>
<td>ICAM O CData</td>
<td>90</td>
</tr>
<tr>
<td>IcamRangeErrorEnable</td>
<td>91</td>
</tr>
<tr>
<td>ICAM Samples</td>
<td>92</td>
</tr>
<tr>
<td>IcamSampleStatus</td>
<td>93</td>
</tr>
<tr>
<td>IM eas</td>
<td>94</td>
</tr>
<tr>
<td>Lmeas</td>
<td>95</td>
</tr>
<tr>
<td>M Kelvin</td>
<td>97</td>
</tr>
<tr>
<td>M M ux</td>
<td>98</td>
</tr>
<tr>
<td>OpenPinCheck</td>
<td>100</td>
</tr>
<tr>
<td>Test Description</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>RCM eas</td>
<td>102</td>
</tr>
<tr>
<td>RCM eas2</td>
<td>104</td>
</tr>
<tr>
<td>RCM eas3</td>
<td>106</td>
</tr>
<tr>
<td>Rmeas</td>
<td>108</td>
</tr>
<tr>
<td>Rmeas2</td>
<td>110</td>
</tr>
<tr>
<td>RofRCM eas</td>
<td>113</td>
</tr>
<tr>
<td>RofRCM eas2</td>
<td>115</td>
</tr>
<tr>
<td>RofRCM eas3</td>
<td>117</td>
</tr>
<tr>
<td>ShortsM eas</td>
<td>119</td>
</tr>
<tr>
<td>SKelvin</td>
<td>121</td>
</tr>
<tr>
<td>SM ux</td>
<td>122</td>
</tr>
<tr>
<td>TPol</td>
<td>123</td>
</tr>
<tr>
<td>TwoSrcDM eas</td>
<td>125</td>
</tr>
<tr>
<td>TwoSrcRM eas</td>
<td>127</td>
</tr>
<tr>
<td>UnGuard</td>
<td>129</td>
</tr>
<tr>
<td>UnGuardBd</td>
<td>130</td>
</tr>
<tr>
<td>UnVSrcBd</td>
<td>131</td>
</tr>
<tr>
<td>VM eas</td>
<td>132</td>
</tr>
<tr>
<td>VSrcBd</td>
<td>133</td>
</tr>
<tr>
<td>ICAM Timing Issues</td>
<td>134</td>
</tr>
<tr>
<td>Test Descriptions</td>
<td>137</td>
</tr>
<tr>
<td>AltSrcRM eas</td>
<td>140</td>
</tr>
<tr>
<td>BetaM eas</td>
<td>141</td>
</tr>
<tr>
<td>CDischarge</td>
<td>142</td>
</tr>
<tr>
<td>Capacitor Discharge Procedure</td>
<td>143</td>
</tr>
<tr>
<td>CM eas</td>
<td>145</td>
</tr>
<tr>
<td>CPol</td>
<td>146</td>
</tr>
<tr>
<td>CofRCM eas</td>
<td>147</td>
</tr>
<tr>
<td>CofRCM eas2</td>
<td>148</td>
</tr>
<tr>
<td>CofRCM eas3</td>
<td>149</td>
</tr>
<tr>
<td>DVM eas</td>
<td>150</td>
</tr>
<tr>
<td>DZM eas</td>
<td>151</td>
</tr>
<tr>
<td>LM eas</td>
<td>152</td>
</tr>
<tr>
<td>OpenPinCheck</td>
<td>153</td>
</tr>
<tr>
<td>RCM eas</td>
<td>154</td>
</tr>
<tr>
<td>RCM eas2</td>
<td>156</td>
</tr>
<tr>
<td>RCM eas3</td>
<td>157</td>
</tr>
<tr>
<td>RM eas</td>
<td>159</td>
</tr>
<tr>
<td>RofRCM eas</td>
<td>160</td>
</tr>
<tr>
<td>RofRCM eas2</td>
<td>161</td>
</tr>
<tr>
<td>RofRCM eas3</td>
<td>162</td>
</tr>
<tr>
<td>ShortsM eas</td>
<td>163</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>DCI_f</td>
<td>195</td>
</tr>
<tr>
<td>DelayC_f</td>
<td>196</td>
</tr>
<tr>
<td>Event_f</td>
<td>196</td>
</tr>
<tr>
<td>Fixture_f</td>
<td>197</td>
</tr>
<tr>
<td>IArly_f</td>
<td>197</td>
</tr>
<tr>
<td>IC_dig_f</td>
<td>198</td>
</tr>
<tr>
<td>IM_rly_f</td>
<td>198</td>
</tr>
<tr>
<td>IPower_Mon_f</td>
<td>200</td>
</tr>
<tr>
<td>IRail_f</td>
<td>201</td>
</tr>
<tr>
<td>IRDACs_f</td>
<td>201</td>
</tr>
<tr>
<td>Meas_F_f</td>
<td>202</td>
</tr>
<tr>
<td>Meas_I_f</td>
<td>202</td>
</tr>
<tr>
<td>Meas_V_f</td>
<td>203</td>
</tr>
<tr>
<td>MRLY_dig_f</td>
<td>204</td>
</tr>
<tr>
<td>SampleC_f</td>
<td>204</td>
</tr>
<tr>
<td>SHFMux_f</td>
<td>204</td>
</tr>
<tr>
<td>XO_Phase_f</td>
<td>205</td>
</tr>
<tr>
<td>Appendix A - Wirelist Import Format</td>
<td>207</td>
</tr>
<tr>
<td>AutoGen Import Description Rev. 0.00</td>
<td>208</td>
</tr>
<tr>
<td>Appendix B - AutoGen CAE Wirelist Format</td>
<td>215</td>
</tr>
<tr>
<td>CAE Wirelist Format Definition For AutoGen</td>
<td>216</td>
</tr>
<tr>
<td>Appendix C - Sample.frm</td>
<td>227</td>
</tr>
<tr>
<td>CYX32.frm</td>
<td>228</td>
</tr>
<tr>
<td>Appendix D - agresult.bas</td>
<td>243</td>
</tr>
<tr>
<td>agresult.bas</td>
<td>244</td>
</tr>
<tr>
<td>Appendix E - Sample Test Techniques</td>
<td>253</td>
</tr>
<tr>
<td>Diode Bulk Resistance Measurements</td>
<td>254</td>
</tr>
<tr>
<td>FET N-Channel, P-Channel, Enhancement Mode or Depletion Mode Open</td>
<td>254</td>
</tr>
<tr>
<td>Resistance and Closed Resistance Measurement</td>
<td>255</td>
</tr>
<tr>
<td>Relay Coil Resistance, Open Contact Resistance, and Closed Contact</td>
<td>259</td>
</tr>
<tr>
<td>Resistance Measurements</td>
<td>259</td>
</tr>
<tr>
<td>SCR Forward Bias Voltage Drop Test</td>
<td>261</td>
</tr>
<tr>
<td>Appendix F - Digalog Error Codes</td>
<td>263</td>
</tr>
<tr>
<td>Digalog Error Codes</td>
<td>264</td>
</tr>
<tr>
<td>Appendix G - Glossary Of Terms</td>
<td>265</td>
</tr>
</tbody>
</table>
System Description
The In-Circuit Analog Measurement (ICAM) board and associated software make it possible for the Series 2040 Test System to perform analog in-circuit testing. ICAM can be used as a stand-alone application for MDA testing, or it can be used to compliment the functional test capabilities of the Series 2040 Test System to provide a fully integrated combinational tester.

Hardware Overview
The ICAM board includes the static, ramped, and sinusoidal voltage and current sources, combined with a multirange measurement system covering AC and DC current, voltage, time, and phase measurement capability to cover the complete range of analog in-circuit measurements. These include guarded measurement of shorts, opens, resistance, capacitance, inductance and semiconductor junctions (e.g. diodes).

The ICAM board can be used with up to sixteen 64 x 4 Matrix Relay (MRLY) boards which multiplex the source, measure, and guard channels for up to 1024 vacuum fixture pins for analog in-circuit testing. The MRLY board, with its 200 volt rating, is also designed for use in the Functional Test Systems.

When the Analog In-Circuit and Functional Test modes are combined in the Series 2040, the 64 Channel MRLY boards are also used to matrix signals to the Testhead measurement system. The ICAM board has four multiplexers which provide the necessary interface for the 64 channel MRLY boards to connect to the input multiplexers of the Analog Measurement System board. This connection allows the Functional Test System access to every UUT node covered by the MRLY boards in the in-circuit system.

Software Overview
Analog In-Circuit testing is done using Digalog’s Windows® based AutoGen software. This software provides for automatic test generation, including calculation of guard requirements. This is accomplished by using net and part list information obtained from the user’s CAD system (see Appendix A) or entered manually. Parts Editor and Net Editor windows allow easy manual editing. The Part Editor window also provides an interactive manual mode to speed test debugging. A graph mode is included to display measurement results. This mode quickly exposes the effects of circuit conditions to speed the debugging of analog tests for isolation of manufacturing defects.
AutoGen Environment

The AutoGen Environment screen, shown below, accesses the features of AutoGen that control automatic test generation, editing, and debugging.

Parts

The initial screen is split into three Windows style tab folders for a parts editor, a net editor, or a Visual Basic editor. The parts editor contains two “sub tabs” to Debug the tests, or edit the test Parameters.

Debug - This folder is used to setup and debug the tests on each individual part in the project. After the CAD list is selected, the Debug screen can be used to set individual test parameters for each part. The “Iterator” can be used to repeat and fine tune each test sequence for optimum performance. The results of each sequence are also displayed showing information on any failures incurred in the Log Window.

Graph - This dialog can be selected to display the output of the A/D converter as shown to the right. This is used to view the stability of the output of the test. The graph displays measurements vs. the number of samples taken over a period of time. This criteria can be used to determine the length of the test.
Parameters - This dialog is used to check or modify the global test parameters for each different type of test. It is also used to assign MUX channels. In addition, the tests for all of the applicable parts in the parts list are generated using this dialog.

Nets

When this tab is selected, a different dialog is displayed (shown on page 12) showing net information for all of the nets in the CAD list. Initial data for the Nets dialog is typically translated from the downloaded CAD list, but may be changed or altered through manual editing. Part-Pins displays net and parts information. Shorts Test is used to generate the shorts testing, and Assign Channels is used to determine MUX channels. When an individual net is selected from the list on the lower portion of the dialog, a parts list for that net is displayed and part information for each individual part can be viewed in the upper right of the dialog as shown on the next page.
Part-Pins - In the above example, net N00035 is highlighted, part R60 is referenced under the Part Pin Editor, and other nets connected to this part are displayed to the right of the dialog under the Part Pin List.

Shorts Test - When this tab is selected, the dialog below is displayed showing all of the test parameters. This dialog is also used to generate the shorts test parameters. When the shorts test is run, the results are displayed in a textbox at the bottom of the main form. The right section of this dialog is used to verify the test fixture, and will be discussed later.
Assign Channels - This dialog is used to assign multiplexer channels to the nets. The base (starting) channel is usually channel #0 but may be changed using this dialog. Panel Parameters is used to designate which board is being programmed when multiple boards of the same type are being tested on the same fixture.

Basic

When this tab is selected, a Visual BASIC immediate window is opened for the creation and declaration of User tests.

CYX Executive API

Digalog Systems has created a new test executive interface for executing the tests generated by the AutoGen program as shown on the next page. The code for this interface is generated by selecting the Generate Executive option from the AutoGen File menu, and can be called up from Visual Basic.

The program simply includes Start and Stop command buttons to execute or halt the execution of the tests generated by the AutoGen routine. Any failed tests are displayed in the results text box and may be viewed using the scroll bars provided. The data can also be sent to a printer or logged in a data file.

The code generated by AutoGen may be used as a simple Test Executive, or it may be modified for the user’s specific needs. The code may also be used with an existing user executive routine. The interface includes a list of functions, variables, and Visual Basic classes, some of which can be changed to suit the user’s needs. This API may also be used with the Cyclops program, and includes the following categories:
Changeable code
Unchangeable code
AutoGen generated code
Cyclops generated code

A detailed explanation of the code is covered under the CYX Executive API section on page 44.

Power Supplies
In addition to the normal Series 2040 Test System power supply voltages, the ICAM board requires ± 40 VDC for operation. These additional voltages are generated by power supplies located in the Testhead area to the right of the Testhead (as viewed from the front of the tester). The + 40 VDC supply is the one located towards the rear of the Testhead area, and the -40 VDC supply is located towards the front. A pin out of both supplies is shown to the right.

Vacuum Control System
The Model 170 & 180 Vacuum Control Systems from Digalog are complete self-contained systems for providing vacuum control to test fixtures for analog in-circuit testing. Model 170 is a single well vacuum controller, and the Model 180 is a dual well controller. Both systems feature a RS-232 and an IEEE-488
port for external communications. A diagram of the Model 180 system is shown below. Note, both systems have self-contained power supplies and microcomputer-based controller boards. These systems are capable of air flow around 290 CFM and provide quick vacuum pull down even in the presence of small leaks. Additional information on the vacuum control systems can be obtained by reviewing the Vacuum Control System manual (PN # 4200-0130).
Installation
Installation

As with any other Windows® installation, it is recommended that you close all other applications before installing the AutoGen software. The software may be installed by inserting Disk #1 in Drive A and running A:\Setup.exe from the Run option from the Start menu, or by double-clicking on Setup.exe on Drive A from the Windows Explorer. When this is performed, the dialog below is displayed explaining the installation. The actual appearance of the dialog may vary with operating systems, but the prompts and messages will be the same. Click “Next >” to continue.

The setup program will then prompt for a destination directory. The default is C:\\Digalog. However, the software can be installed on any drive with enough open memory space. Since other Digalog software will search for the “Digalog” directory, it is recommended that the installation directory for this software be named “Digalog.” When this is selected, the dialog shown on the next page is displayed with the option for backing up existing files. Obviously, a first time installation would not require a backup. If this is an upgrade from a previous version of AutoGen software, you may want to backup the current files. If “Yes” is selected, a standard dialog appears prompting for the name and location of the directory for the backup files.
When this is completed, the prompt below is displayed to determine if the software is being installed on a 2040 system or just a stand-alone development computer. When this selection is completed, two dialogs are shown prompting for the name and location of the program group for this installation. In most cases, use the default name and location. Once completed, the setup program begins installing the AutoGen V4.0 software. When the installation is complete, the program group will be displayed showing icons for all of the software installed.

Before running the AutoGen software, the computer must be restarted. In addition, the Tester Resource Manager, TRMAN, must be run in order to store the target 2040 tester configuration in the Registry. If not, a “Runtime Error 2” will be displayed.
The installation software adds a line to the autoexec.bat file setting the “Digalog” environment variable to C:\Digalog. However, some newer operating systems may require that this environment variable be manually set using the System\Advanced\Environment option from the Control Panel.

NOTE: Since Digalog is constantly striving to provide the latest and most complete technology and techniques available, updates to the AutoGen software will be made available in the form of Service Packs. The latest available Service Packs should be installed in the same manner as the original software. If you are unsure about the latest available Service Pack, contact the Digalog Customer Service Department at (262) 785-8777.
All programming for the In-Circuit Analog Measurement System is done through Digalog's AutoGen software. This software uses Windows® as an operating system. To open the AutoGen program, use the Windows Explorer or create a shortcut on the Desktop. When the program is selected, the dialog shown below appears.
File Menu/Toolbar

The AutoGen File menu contains the standard Windows options for a New project, Open an existing project, Save the current project, Save the current project As another filename, and Exit. In addition, options are provided to Generate Executive, Hardware, Import, Create Wirelist, and Print.

**Generate Executive** - This option creates a Test Executive for the current project.

**Hardware** - When this option is selected, a small dialog appears allowing the user to select which of the available Matrix Relay boards are used for Measurement, which are used for Kelvin, and which are not in use. The graphic indicates that boards #0 through #2 are used for Measurement, and #3 is used for Kelvin. This configuration is used by the “Assign Net/Mux channels” dialog discussed on page 34. Up to a total of 16 boards may be present in the system.

The Fixture ID section is used to display the current Fixture ID number. When a new project is created, use the Get Fixture ID command button to instruct AutoGen to get the current ID number. If no fixture is present on the system (development only), the proposed Fixture ID may be entered in the textbox with the Windows text tool.

**Import** - This option displays a common dialog for importing a “*.net” wirelist into the project.

**Create Wirelist** - This option allows the user to create a wirelist for the construction of a test fixture.

**Print** - This option allows the user to print All Parts tests, just Tested parts, just Untested parts, just the Disabled parts, or the Wirelist. Disabled parts are parts purposely disabled by the programmer due to specific
testing problems. Untested parts include disabled parts and all parts whose component type is set to “Other”. It does not include “User” tests that do not contain any code.

If New or Open is selected from the menu, a small listbox appears prompting the user to select from a list of current projects in the Projects directory or enter the name of a new project. If a new project is selected, a wirelist may be imported into the project. If any errors occur during the import process, a log file is displayed in a text box on the bottom of the main dialog listing the errors. Viewing and saving this log file will be discussed under the Edit Menu.

**Edit Menu**

**Parts** - When this option is selected, a small menu is displayed with options to insert a new part before the currently selected part, after the currently selected part, or to delete the currently selected part. If the Part Editor is not currently displayed, i.e. another folder is selected, this menu will appear ghosted and is not active.

**Pins** - When this option is selected, a small menu is displayed to add a pin, delete a pin, toggle the current pogo pin, or assign a free pin as a pogo pin. Again, if the Part Pin Editor is not currently displayed, i.e. another option folder is selected, this menu will appear ghosted and is not active.

**Nets** - When this option is selected, another small menu is displayed to insert a net before the currently selected net, insert a net after the currently selected net, or delete a net. If the Net Editor is not displayed, this menu appears ghosted.
**Log Window** - This option is used to toggle the Log Window at the bottom of the main form on or off, or to save the current log to a file. If the log is to be saved, a common dialog is displayed for a name and location for the log file.

**Find & Replace** - This option will display a dialog for finding and/or replacing text. The desired Find/Replace column must be selected, and the Find/Replace text can be entered in the textbox.

**Swap S&M** - This option swaps the source and measure channels for the currently selected part.

**Part Pin 1 Net, Part Pin 2 Net** - These listboxes display the names of the currently designated nets for the source and measure pins. Usually it is used when adding 2-pin parts easily. When a net is changed, all the guards selected in the Guard Editor are cleared. The comboboxes contain a list of all the nets currently defined in the net editor. These should not be used for parts with more than 2 pins.

**Vacuum**

The Vacuum menu is used to select or deselect DUT1 (#1 vacuum well) or DUT2 (#2 vacuum well on units so equipped).

**Execute Menu**

**Execute** - This option executes the part test range indicated in the **Start Test** textbox and the **End Test** textbox for the number of iterations selected in the **Iterations** textbox (explained below).

**Execute All** - This option fills the **Start Test** and **End Test** textboxes with the first and last parts respectively and executes all of the tests for the number of iterations selected.

**Stop** - This option halts the execution of the test sequence.

**Continue** - This option resumes execution of a test sequence after the sequence has been “Stopped” or “Stop On Failure” has been selected. If “Track Current” has been selected also, the “**Start Test**” textbox is
changed to the failed test, and “Continue” resumes execution to the End Test.

**Graph** - This option executes the currently selected test and takes 10 samples over the designated test time. A graph of the results is then displayed. **Calibration offsets are not used in displaying the graph. The graph is used only to determine test stability.**

**Calibrate** - This option samples all parts tests designated by the “Use” flag under the Calibration column with the vacuum fixture up, and then sets zero offsets for these tests.

**Start Test** - This textbox contains the number of the currently selected test. The Text Tool may be used to change the Start Test by highlighting the current value and typing the desired value.

**End Test** - This Textbox contains the number of the last test in a selected sequence, or it can contain the same test number as the Start Test when a specific test needs to be run a number of times and checked for stability.

**Iterations** - This textbox contains the number of iterations for a single test or group of tests for a determination of test stability.

**Standard Deviation Mode** - When checked, a statistical analysis will be performed on the iteration data collected during the execution of the selected part or user tests. The mean, standard deviation and percent of range values are calculated and displayed along with the minimum and maximum measured values in the Log Data Window for each of the selected parts.

The population standard deviation equation (n-1) will be used if the number of iterations is less than 30; otherwise, the sample standard deviation will be used. The percent of range calculation may be used to determine the validity of a part or user test. The percent of range value
is defined as ratio of the tester variance to the part tolerance range and is calculated as follows:

\[
\text{percent of range} = \left( \frac{2 \times \text{xstd}}{\text{UL} - \text{LL}} \right) \times 100
\]

Where:

- \( \text{mean} \) = iteration data average
- \( \text{xstd} \) = standard deviation multiplied by the Standard Deviation X Factor
- \( \text{UL} \) = part value upper limit
- \( \text{LL} \) = part value lower limit

A test is determined to be invalid if any one of the following three conditions occur:

1. \( \text{mean} + \text{xstd} > \text{UL} \)
2. \( \text{mean} - \text{xstd} < \text{LL} \)
3. \( \text{percent of range} > 30\% \)

The results will be displayed in the Log Data Window if the Display All Data option is selected or if a test fails.

**Note:** This option will be ignored if the number of Iterations is less than two.

**Standard Deviation X Factor** - This list box contains the number of standard deviations to use in determining the pass/fail criteria of the Standard Deviation Mode. The selectable values of this number range from 1 to 6.

**Validate R or C Tests** - This option validates all selected resistor, capacitor, user resistor or user capacitor part tests. Part validation is accomplished by taking a measurement with and without an IVAL component connected in parallel with the device under test and verifying that the appropriate change in measured values occur. Up to three different IVAL component values may be used for validation in each resistor or capacitor part test. IVAL components will be automatically selected to invoke an expected change of approximately 10%, 25% and 50% of the part value being verified, beginning with a 10% change. The part verification test will continue with the 25% or 50% changes only if the validation test fails with the 10% change.
The Validation Tests calculates a Sensitivity value that is used to determine the validity of a test. This value represents the percentage of the expected measurement change that actually occurred with the IVAL component connected in parallel and is calculated using the following:

\[
\text{\%Sensitivity} = \left( \frac{\text{Actual change}}{\text{Expected Change}} \right) \times 100
\]

A test is considered invalid if the sensitivity exceeds the upper limit or falls below either of the two sensitivity thresholds selected in the %UL, %PV or %PP text boxes. Test results are displayed in the Log Data Window based on the Display All Data selection.

- **% UL** - This textbox contains the upper limit of the Part Value sensitivity (in percent) used in resistor and capacitor test validation. A part validation test will fail if the reported sensitivity exceeds this limit during validation testing. A failure description will also be included with the test results in the Log Data Window.

- **% PV** - This textbox contains the Part Value - Part Present sensitivity threshold (in percent) used in resistor and capacitor test validation. A part validation test will fail as “Part Present” if the reported sensitivity falls below this limit during validation testing. A failure description will also be included with the test results in the Log Data Window.

- **% PP** - This textbox contains the Part Presence - Transparent To Test sensitivity threshold (in percent) used in resistor and capacitor test validation. A part validation test will fail as “Transparent To Test” if the reported sensitivity falls below this limit during validation testing. A failure description will also be included with the test results in the Log Data Window.

**Measure IVAL Components** - This option measures each component on the In-Circuit Validation Board (IVAL). This feature is provided to verify IVAL operation and component values. The measurement results are displayed in the Log Data Window based on the Display All Data selection. NOTE: A fixture must be present to use this option.

**Track Current** - When this option is selected, the currently selected part from the Part Editor or currently selected Net from the Net Editor updates the Start Test and End Test textboxes. If it is not selected, the textboxes can only be updated or changed manually, or by clicking on Execute All. This option is helpful in debugging a range of tests.
**Stop On Failure** - If this option is selected and tests are being executed using the Execute, Execute All, or Continue options, the first failure will stop execution of the sequence. If not selected, the sequence will continue to the End Test and all iterations will be performed.

**Display All Data** - When this option is selected, all test data (pass or fail data) is sent to the Log Data Window. If it is not selected, only failed data is sent. (NOTE: this does not apply to the Shorts and Opens Tests where all data is sent to the Log Data Window regardless of the Display All Data option.)

**Save All Data** - When checked, all part or user test execution data will be save to a file after all of the selected tests have run. The file is formatted so that the iteration data for each test is Tab delimited for easy import into a spreadsheet.

**Help**

The Help menu contains the standard Windows options for Contents, Search, and an About window with the software version levels.

**Toolbar**

The Tool Bar contains shortcut buttons for most of the options from the various menus of the Menu Bar. When the options are active (displayed), the associated tools will appear in color. When the options are inactive, the tools will appear ghosted. To determine the function of each tool, place the mouse cursor over the desired tool for a second or two and a small prompt appears with the function of that tool.

**Matrix Grid**

A row/column matrix grid format is used to display the Parts Editor, Guard Editor, Net Editor, Part Pin Editor, and Part Pin List. The grid rows contain the Part, Net, or Guard reference data organized in columnar form. The intersection of the row and column forms a cell which contains specific data for the selected Part, Net, or Test. Editing can be accomplished using the keyboard arrow keys, under mouse control, or a combination of both.

**Keyboard Mode** - The up/down arrow keys are used to select a particular row; the left/right arrow keys to select the desired column. Selected position is indicated by a highlighted cell border. Data is entered from the keyboard with the first keystroke (other than an arrow key) erasing the existing data, and displaying the keystroke character.
Left/right arrow keys can now be used to position cursor within the cell.

Once the blinking cursor appears in the cell, the text is ready to be edited. The up/down arrow keys will enter data and reposition the selected cell one cell up or down. The tab key will enter data and reposition the selected cell one cell to the right. The return key will enter data without change in position. Once data is entered, the left/right arrow keys again move column selection to left or right.

**Mouse Mode** - The mouse is used to position the cursor at a desired cell position. The first click of the left hand button selects the cell (same indication as above). The second click inserts the Windows text tool for data entry but does not erase existing data. The mouse may be used to reposition cursor within field for editing. Backspace and delete keys are active within cell. Data may be entered by using cursor to select another cell, or using the keystroke mode options above since mouse and keystroke mode operation can be intermixed.

When the right hand mouse button is clicked (on the Parts Grid), a small dialog appears with the options to **Insert Part After**, **Insert Part Before**, **Delete Part**, or **Find/Replace**. Similar options are displayed when the right hand mouse button is clicked on the Net Grid. When the Part Pin Grid is edited in this manner, the second small dialog to the right is displayed with options to **Add Part Pin**, **Delete Part Pin**, **Toggle Pogo Pin**, or **Assign Free Pogos**. **Toggle Pogo Pin** will toggle the pogo pin between the current pogo pin and the selected pogo pin (if different), or add a pogo to the currently selected pin if no pogo pin was designated. **Assign Free Pogos** will assign a pogo to the first pin in the current list if a pogo pin was not designated.

**Matrix Grid Format Changes**

The matrix grid may be modified during an editing session using click and drag method. The modified grid format will be retained until the current AutoGen program session has been closed.

**Column Width Change** - The cursor is positioned on the border to be changed (proper location indicated by cursor change to double arrow) then clicked and held. Dragging the cursor to the right will expand the column to its left by displacing right-hand columns further right. Dragging the cursor to the left will decrease column width by moving right-hand columns to the left. Dragging the right-hand border to the
left-hand border will temporarily delete the column.

The column modifications will be maintained during the editing session until the current AutoGen session is exited. Data called up in next AutoGen session will be displayed in standard matrix grid format.

Parts

Part Editor

This area of the dialog contains a listing of every part in the project. Parts may be added or deleted using the Edit menu from the menu bar (or the dialog displayed with a right button mouse click). When parts are added, all information should be entered or errors can occur when the tests are generated. Each column is briefly explained below.

Test # - The number of the specific test determined by the location of the part in the grid.

Reference - The reference number of the part, usually determined by the schematic or net list.

Name - Type or value of the part. Sometimes the industry part number is used.

Number - The manufacturer or vendor part number.

Type - The type of test to be run.

Src - The Matrix Relay channel to which the source will be applied.

Meas - The Matrix Relay channel used as a measure channel for the test.

Kelvin - The Kelvin mode for the test. Kelvin modes can be “None”, “Source”, “Measure”, or “Source & Measure”.

Forcing Val - The forcing value used for the forcing function. These values should vary depending on the value of the component.

Limit - Maximum value applied for the test type as set in the Parameters “subtab”.

An example of the default limits are as follows:
### Table

<table>
<thead>
<tr>
<th>Test Type</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resistance</td>
<td>300ma</td>
</tr>
<tr>
<td>Capacitance</td>
<td>300ma</td>
</tr>
<tr>
<td>Inductance</td>
<td>300ma</td>
</tr>
<tr>
<td>Diode V</td>
<td>3.00 Volts</td>
</tr>
<tr>
<td>Zener</td>
<td>3.00 Volts</td>
</tr>
</tbody>
</table>

**Time/Freq** - Duration of the test or frequency of the forcing voltage (Inductor Test). The graph mode can be used to determine a stable time for each individual test.

**Exp Value** - The actual value of the component being tested.

**+ Tol** - Upper limit as a percentage for expected values.

**- Tol** - Lower limit as a percentage for expected values.

**# Samp** - The number of test samples to be taken. If this number is greater than 1, the test is executed as many times as shown and an average is taken and returned. If this number is greater than 10, the test is executed as many times as shown and the highest and lowest readings are removed. An average is calculated and returned. This is not done for user tests, but the value is passed into the user test through the Part Data “Number Of Samples” property.

**Calibration** - This column is only enabled for Capacitance components. It consists of a command button that toggles from “Use” to “Don’t Use”. All other component entries are shown as N/A for (Not Applicable). When “Don’t Use” is selected, nothing is done. When “Use” is selected, the measured calibration (generated by clicking the “Calibrate” button on the Test Information frame) is subtracted during the execution of the capacitance test and the new value is checked against the tolerance of the component.

When the code is generated for CYX, code is added to calibrate the fixture in the “up” position before the vacuum is enabled. When vacuum is applied to the fixture (down), the calibration value is subtracted from the measured value for every capacitor component that was selected as “Use”.

**Discharge** - This column is available for every enabled component. Like the Calibration column, it consists of a command button that toggles from “Use” to “Don’t Use”. Disabled component entries are...
shown as N/A for (Not Applicable). When “Don’t Use” is selected, nothing is done. When “Use” is selected, a delay equal to the test time is executed after the measurement is taken but before any channels (i.e. source channel, measure channel, kelvin channel(s), and/or guard channels) are disconnected from the part. This allows any charge on the capacitors to dissipate through the source channel. The discharge feature allows for more accurate measurements and prevents harmful current spikes. This code is also included in the generation CYX code.

**Mode** - Determines whether the test is enabled or disabled.

**Comments** - Special comments or conditions for each individual test.

**NOTE:** Each of these parameters may be manually edited to suit the particular needs of an individual test. The values listed in the Parameters “subtab” are default values.

**Graph**

With the Debug subfolder selected, a graph will be displayed similar to the graphic shown below. When the Graph option is selected (using a the shortcut tool or the Execute Menu), a graph of the currently selected part test is displayed (if the test is a valid test and enabled). Calibration values are not used in displaying the graph. The primary function of the graph is to determine the stability of the test.

![Graph](image)

The test time is segmented into 10 equal samples and the x-axis on the graph shows the sample number. There are always 10 samples taken for any test. So, for example, if there is a 1 millisecond test time, each sample is 100 microseconds. Also, if there is a 10 millisecond test time, each sample is 1 millisecond. Therefore, the graph will always show the entire test on the graph in full scale (for samples).
The y-axis of the graph is the component value being read at a specific sample. If the component type is a capacitor, the units are farads. If the component type is a resistor, the units are ohms. If the component type is a diode voltage, the units are volts. The values on the y-axis are autoscaled for the period that the samples are taken. During real testing, there is only 1 sample taken at 90% of the test time. The graph results at 90% and the actual test results should be approximately the same value.

**Guard Editor**

With the **Debug** subtab selected, a grid for the Guard Editor is displayed. A typical guard list is shown below. The Guard Editor is used to maintain a list of guards for the currently selected part. Guards can be added and deleted from the guard list via the Guard menu. This menu can be displayed by right-clicking on the Guard Editor or the proper tool from the Tool Bar may be used.

<table>
<thead>
<tr>
<th>Net</th>
<th>Chan</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGND_0062B</td>
<td>15</td>
</tr>
</tbody>
</table>

**Net** - Net name of existing guards or the guard added. This field is not editable but can be selected using the Chan column.

**Chan** - This column is used to select the channel number (from the current board selected in the Panel Parameters Frame).

**Note:** This is the only place where guards may be assigned.

**Parameters**

When this subtab is selected, a dialog appears containing two frames for AutoGen Parameters and Test Parameters.

**AutoGen Parameters** - This is the mode control section of the Debug tab and includes the following:

**Assign Mux Channels** - Instructs the program to automatically assign Mux channels for new parts or parts or nets that have moved or changed.

**Replace Mux Channels** - Instructs the program to reassign all mux channels. This option is not recommended after fixture.
wiring has been defined.

**Calculate Test Parameters** - Instructs the program to automatically generate test parameters based on current Test Parameter ranges and limit values. Any disabled test that is now within parameter range will be added as an enabled test.

**Note: No enabled tests are generated.**

**Replace Existing Params** - Causes all tests to be replaced with the new set of values calculated during the next auto generation cycle including enabled tests.

**Generate Tests** - When all of the modes and Test Parameters have been set, the Generate Tests command button will automatically generate new tests for the project.

**Test Parameters** - This frame contains controls for setting the upper and lower range limits for the parameters of each type of test. A listbox is provided to select the parameter to be set. The following table gives the Digalog default values for these parameters. These values have been set at typical values for normal test situations.

The hardware default values give the maximum values for the tester hardware. All other default values are given for typical test situations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Source Max</th>
<th>Source Min</th>
<th>Measure Max</th>
<th>Measure Min</th>
<th>Test Time Max</th>
<th>Test Time Min</th>
<th>Limit Max</th>
<th>Limit Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resistance</td>
<td>200mV</td>
<td>50mV</td>
<td>1mA</td>
<td>4uA</td>
<td>100mS</td>
<td>100mS</td>
<td>100mA</td>
<td>100mA</td>
</tr>
<tr>
<td>Capacitance</td>
<td>8KV/S</td>
<td>50V/S</td>
<td>750mA</td>
<td>4uA</td>
<td>100mS</td>
<td>1mS</td>
<td>300mA</td>
<td>300mA</td>
</tr>
<tr>
<td>Diode V</td>
<td>1mA</td>
<td></td>
<td></td>
<td></td>
<td>1mS</td>
<td>3.00V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diode I</td>
<td>1Volt</td>
<td></td>
<td></td>
<td></td>
<td>1mS</td>
<td>300mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inductance</td>
<td>200mA</td>
<td>50mA</td>
<td>750mV</td>
<td>4uA</td>
<td>100mS</td>
<td>1mS</td>
<td>300uA</td>
<td>300uA</td>
</tr>
</tbody>
</table>
Default values may be changed within the limits of the tester hardware. These values are global values stored in the Registry. The values selected will be the default values retained and used for the generation of all new tests.

Nets

When this tab is selected, the dialog changes to the net editor screens as shown below. The Menu Bar for this section of AutoGen is the same as the Parts section except that the Edit menu deals with nets and not parts.

Net Editor

The Net Grid contains the connectivity relationship between board parts. Initial data for the grid is typically translated from the downloaded CAD wire or net list. This information may be changed, added to, or deleted through manual editing. The net list file is also used to develop the test for shorts
between nets or nodes. (Also see Panel Parameters on page 41.)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Net Name</strong></td>
<td>The reference name or number used to identify the individual nets.</td>
</tr>
<tr>
<td><strong>Pogo Part Reference</strong></td>
<td>Specific part on each net designated as the Pogo part. This part is selected in the Part-Pin Editor.</td>
</tr>
<tr>
<td><strong>Pogo Pin Name</strong></td>
<td>Specific pin on the Pogo part designated as the pogo pin. This pin is selected in the Part-Pin Editor.</td>
</tr>
<tr>
<td><strong>Shorts Threshold</strong></td>
<td>Two nets are considered shorted together if the resistance between them is below this threshold.</td>
</tr>
<tr>
<td><strong>Opens Threshold</strong></td>
<td>Two nets are considered open if the resistance between them is above this threshold.</td>
</tr>
<tr>
<td><strong>Test Time</strong></td>
<td>This time may be manually edited for nodes that exhibit an unusually high capacitance value.</td>
</tr>
<tr>
<td><strong>Test Type</strong></td>
<td>This column contains the specific shorts/opens test type for the net. A drop-down listbox for selecting the shorts/opens test type with the mouse is also provided.</td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td>Special notes or comments about this specific net.</td>
</tr>
<tr>
<td><strong>Board 1 Channel</strong></td>
<td>The Mux channel number assigned to the net on board #1.</td>
</tr>
<tr>
<td><strong>Board 1 Kelvin</strong></td>
<td>The Mux channel assigned to handle the additional source and measure sense leads needed for Kelvin connections on board #1.</td>
</tr>
<tr>
<td><strong>Board 2 Channel</strong></td>
<td>The Mux channel number assigned to the net on board #2 if the fixture has a second board.</td>
</tr>
<tr>
<td><strong>Board 2 Kelvin</strong></td>
<td>The Mux channel assigned to handle the additional source and measure sense leads needed for Kelvin connections on board #2 if the fixture has a second board.</td>
</tr>
<tr>
<td><strong>Board 3 Channel</strong></td>
<td>The Mux channel number assigned to the net on board #3 if the fixture has a third board.</td>
</tr>
<tr>
<td><strong>Board 3 Kelvin</strong></td>
<td>The Mux channel assigned to handle the additional source and measure sense leads needed for Kelvin connections on board #3 if the fixture has a third board.</td>
</tr>
</tbody>
</table>
**Part Pin Editor**

When a specific net is selected from the Net Grid, this Part Pin Grid displays a list of all of the parts and the pins of each one of those parts connected to the net. It is also used to designate which part pin on the net is the pogo pin.

In the example to the right, part D36 is highlighted. The Anode (Pin 1) of D36 is connected to Net COM (the net selected from the Net Editor) and is designated as the pogo pin for the net.

**Reference** - The name of a specific part.

**Pin Name** - The name (or number) of the specific pin tied to the selected net.

**Pin #** - Number of the pin tied to the net.

**Pogo** - Shows which part pin on the net is designated as the pogo pin.

### Part Pin List

When a specific part is selected from the Part Pin Editor, the Part Pin List displays a list of all of the nets connected to that part, and the specific pin of the part connected to the net is displayed.

**Net Name** - Name of the specific net.

**Pin Name** - The name of the pin of the selected part tied to the net.

**Pin #** - Number of the part pin tied to the net.

In this example, part D36 was selected from the Part Pin Editor, and both nets connected to D36 are displayed under the Part Pin List. If a net containing an IC (such as U4) was selected from the Net Editor, the IC pin connected to the net would be displayed under the Part Pin Editor. If the IC pin was selected from the Part Pin Editor, all nets connected to a pin on the IC would be displayed.
Thus, a net may be selected from the Net Editor, and the parts tied to the selected net are displayed in the Part Pin Editor. By selecting one of these parts, the programmer can view all of the other nets that the part is connected to in the Part Pin List.

**Note:** By selecting a Net Name from the Part Pin List, the selected net is now highlighted in the Net Editor, and all of the parts tied to that net are now displayed in the Part Pin Editor. Therefore, nets may be selected in both places.

**Shorts Test**

This subtab allows a test of all activated nets for net to net shorts. The test is performed by sequentially measuring the resistance between an individual net and all the other nets.

**Shorts Test Parameters**

The conditions for the test are setup using the Shorts Test Parameter functions, as shown to the right.

**Source Voltage** - Value of the source voltage to be used during the test. Default value is 300 mV.

**Default Test Time** - Establishes the default test time for the test.

**Current Limit** - Establishes the default current limit for the test.

**Source Impedance** - Sets the source impedance limit for the test.

**Max Shorts Resistance** - Used to establish the resistance value below which a net short condition is said to exist. If the calculated Net impedance is greater than this value, this value is used instead.

**Min Opens Resistance** - Used to establish the resistance value above which a net open condition is said to exist. If the calculated Net impedance is less than this value, this value is used instead.

**Shorts Value %** - Percentage of calculated parallel resistance to be subtracted when setting the maximum resistance for a specific shorts test.
**Opens Value %** - Percentage of calculated parallel resistance to be added when setting the minimum resistance for a specific opens test.

**Min Shorts Threshold** - Calculated values below this default will disable the shorts test for the corresponding nets.

When the Generate Test Parameters command button is selected, AutoGen calculates the parallel resistance and the sum of the capacitance for every individual net in the project with the Test Type designated as “Shorts&Opens”. From this calculated value, AutoGen subtracts the “Shorts Value %” to determine the maximum resistance for a short, and AutoGen adds “Opens Value %” to determine a minimum resistance for an open. If the maximum resistance for a short is greater than the default **Max Shorts Resistance**, the default is used during the shorts test. If the minimum resistance for an open is less than the default **Min Opens Resistance**, the default is used during the opens test. If the maximum resistance for a short is less than the **Min Shorts Threshold**, the shorts test is disabled. Typically, the resistance for the shorts and opens values should be between the shorts and opens thresholds. The test time is determined by the RC time constant based on the parallel resistance and the sum of the capacitance.

For nets with a Test Type designated as either just Shorts or just Opens, no calculations are performed, and the shorts and opens thresholds in the Net Editor will remain unchanged. For nets with high total capacitance, test times for these nets may be increased.

**Verify Test Fixture**

Controls on this frame are used to verify the test fixture. The Probe Channel textbox is used to enter the number of the probing multiplexer channel. This can be any unused multiplexer channel. To check an assigned channel, connect one end of a shorting probe to the Probe Channel and the other end to the assigned channel. When the **Probe** command button is selected, the program performs a shorts test on the system, and the channel and net shorted by the probe to the Probe Channel should appear in the textbox at the bottom of the form. The Net grid should also scroll to the shorted channel. If not, the fixture has a wiring problem.

**Note:** Most fixtures will have a small jack on the side of the frame connected to a Probe Channel for convenience.
Assign Channels

When the Assign Channels subtab is selected, a frame is displayed as shown to the right. This frame is used to Assign tester Mux channels to each of the nets. A textbox is provided to enter the Base Channel as a starting point. Two check boxes are provided to Replace Existing Channels or to Assign Kelvin Channels. If the Replace Existing channels option is checked, all of the channels for all of the nets in the project will be reassigned. If the Assign Kelvin Channels option is checked, any net which contains a part meeting the threshold criteria displayed in the frame will be assigned a Kelvin channel. These Kelvin channels reside on the Matrix Relay boards designated as Kelvin boards in the dialog shown under Hardware on page 23.

Panel Parameters

The Panel Parameters frame is also included under the Assign Channels subtab. This frame is used for applications where multiple boards of the exact same type are to be tested at the same time. The number of boards to be tested can be selected using the spin control adjacent to the “Number of Boards:” label. When this number is changed, the Net Grid columns are updated to reflect the change, and columns are added or deleted as necessary. Using the 110 net project called “Sample”, the number of boards in the project was changed from one to three, as reflected in the graphic below.

<table>
<thead>
<tr>
<th>#</th>
<th>Comment</th>
<th>Board 1 Channel</th>
<th>Board 1 Kelvin</th>
<th>Board 2 Channel</th>
<th>Board 2 Kelvin</th>
<th>Board 3 Channel</th>
<th>Board 3 Kelvin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>-1</td>
<td>111</td>
<td>-1</td>
<td>222</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</td>
<td>-1</td>
<td>112</td>
<td>-1</td>
<td>223</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2</td>
<td>-1</td>
<td>113</td>
<td>-1</td>
<td>224</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>3</td>
<td>-1</td>
<td>114</td>
<td>-1</td>
<td>225</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>4</td>
<td>-1</td>
<td>115</td>
<td>-1</td>
<td>226</td>
<td>-1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>5</td>
<td>-1</td>
<td>116</td>
<td>-1</td>
<td>227</td>
<td>-1</td>
</tr>
</tbody>
</table>

The columns labeled “Board 2 Channel”, “Board 2 Kelvin”, “Board 3 Channel” and “Board 3 Kelvin” were added to reflect the change. When the Assign channels command button is selected, the program assigns the next
contiguous block (111 - 221) to the second board added with the spin control. Kelvin channels are assigned in the same manner. If the system runs out of Mux or Kelvin channels, a text box is displayed showing which nets could not be assigned a channel.

The “Current Board” spin control is used to select which board in the project (if multiple boards exist) will be used for the shorts tests and parts tests. In this manner, each board can be diagnosed or debugged individually.

**NOTE:** The program assumes that each of the boards in the project is identical to all other boards in the project. Thus, any changes made to a net or part for one board is reflected on the same net or part for every other board in the project.

**Basic**

When this subtab is selected, AutoGen opens a Visual Basic window for the creation of user tests as shown below. There are several steps in creating User
tests for a part within the Part Editor:

1. The part Component Type should be set to User.
2. The part test Mode must be enabled.
3. A valid Visual Basic function name must be entered into the Comment field.

A function is created with the name given in the Comment field in the Basic editor.

The Basic Tab can also have User defined tests for the Shorts and Opens Test. There are several steps in creating a User test for a Net:

1. The net Test Type should be set to User.
2. A valid Visual Basic function name must be entered into the Comment field.

Then two functions are created in the Basic editor: a Shorts function with the Comment field name appended to it and a Opens function with the Comment field name appended to it. (See ShortsMeas in the Test Description Section.)

**Note:** There is a usertest subroutine created (if one doesn’t already exist) called PreGuards() that gets called prior to the guarding of channels such that if certain types of capacitors need to be discharged, the discharge routines may be placed there.

Consult the CYX Executive API section for a description of the PartData class member that gets passed to User functions in the Basic Tab. It contains the properties available to User tests.
CYX EXECUTIVE API

The CYX executive API was designed to enable the use of AutoGen and Cyclops generated code with a user defined executive. The interface is a list of functions, variables, and Visual Basic classes, some of which can be changed to suit the users needs. There are several types of code that the user must know before using the CYX executive API. There are several categories of CYX executive API code:

- Digalog provided changeable code
- Digalog provided unchangeable code
- AutoGen generated code
- Cyclops generated code

**NOTE:** The code generated by AutoGen and Cyclops is to be considered unchangeable.

**Changeable Code**

The CYX executive API contains changeable code provided by Digalog. This code is used as an example of how to interface to the CYX executive API. All of the necessary subroutines, variables, and class instantiations are implemented. The form `project.frm` and `agresult.bas` are changeable.

**NOTE:** Listings for both the `project.frm` (Appendix C) and `agresult.bas` (Appendix D) are given in the back of this manual.

**CYX executive form** - Cyx.frm is the main form where the executive is started. There are two buttons on the form: **Start** and **Stop**.
The Start button starts the test, disables itself and enables the Stop button. The Stop button stops the test, disables itself and enables the Start button. Three text boxes are used for logging of data: one for the screen, one for the printer, and one for the log file. The only option displayed during runtime is the screen. The other two are used only for data storage. There is also a pull down File menu to configure the results and to exit the program. The results for screen, printer and log file can be set to none, fail data or all data. The selections are stored in an `.ini` file so that each time the program is started, it remembers the settings.

When “Generate Executive” is selected from the AutoGen “File” Menu, a Visual Basic test program is produced that integrates the tests produced in AutoGen with the CYX Executive. This creates a user interface for production test. Unless it is changed by the programmer, this VB test program will run the ICAM test program in the following sequence:

- **Shorts Tests**
- **Opens Tests**
- **Component Tests (Standard and User Tests)**
- **Shorts Diagnostics (If Shorts Tests finds more then two and less then 10 Shorts failures)**

The generated test executive also gives the operator the option to output the test data to the screen, printer of a log file.

**Provided Unchangeable Code**

The provided unchangeable code includes four Visual Basic class modules that store information about various tests. These four class modules are:

- **Cyxdata.cls**
- **Partdata.cls**
- **Shrtdata.cls**
- **Agdata.cls**

Each of these are used for cyx tests, parts tests, shorts tests, and cyclops tests.

**CyxData.cls** - The CyxData class is defined in cyxdata.cls and contains the following properties:
Public Name As String
Public PassOp As Integer
Public FailOp As Integer
Public Location As Integer
Public Condition As Integer
Public Test As New Collection

During execution of CYX in the ExecuteCYX() function, this object gets initialized by the values from InitializeCYX(). A copy of the object is made so that each execution of ExecuteCYX() will start off exactly the same way. This is used in root.cls, autogen.cls, shrttest.cls, parttest.cls, and agresult.bas.

PartData.cls - The PartData class is defined in partdata.cls and contains the following properties:

Public Reference As String - This is the part reference designator, usually imported from the project CAD files. This string is found under the “Reference” column in the Part Editor grid.

Public Name As String - The name of the component under test. This usually includes the component value, and is normally imported from the project CAD files. This string is found under the “Name” column in the Part Editor grid.

Public Number As String - This is an in-house part number assigned to a component in the BOM, usually imported from the project CAD files. This string is found under the “Number” column in the Parts Editor grid.

Public ComponentType As Integer - This is the type of component to be tested, designated by selecting the grid cell, clicking on the drop-down arrow, and choosing the desired type. Definitions of the various types for the Visual Basic project executive are found in the Autogen.bas file in C:\Digalog\Include.

Public SourceChannel As Collection - This is the M Rly channel assigned as the source channel for the component under test. It is found under the “SRC” column in the Part Editor grid. It is designated as a collection in order to deal with multiple boards on one panel.
Public **SourceKelvin** As Collection - Source Kelvin channels are created in the Parts Editor. Click on the Kelvin column for the selected part and a drop-down listbox appears with options for None, Source, Measure, and Source & Measure. If None is selected, no Kelvin channels will be selected. If Source is selected, a Source Kelvin channel is assigned to the Source channel of the current part. The assigned source Kelvin channel number is displayed in the Net Editor as a Kelvin channel assigned to the source channel net. For example, resistor R1 has a source channel #5 assigned to its pin-1 (Net 4). If the Source channel option is selected under the Kelvin column, the Net Editor will show that a Kelvin channel has been assigned to Net 4. If the Source & Measure option is selected, a Kelvin channel is assigned to both channels in the same manner. SourceKelvin is designated as a collection in order to deal with multiple boards on one panel.

Public **MeasureChannel** As Collection - This is the MRly channel assigned as the Measure channel for the component under test. It is found under the “MEAS” column in the Part Editor grid. It is designated as a collection in order to deal with multiple boards on one panel.

Public **MeasureKelvin** As Collection - Measure Kelvin channels are created in the Parts Editor. Click on the Kelvin column for the selected part and a drop-down listbox appears with options for None, Source, Measure, and Source & Measure. If None is selected, no Kelvin channels will be selected. If Measure is selected, a Measure Kelvin channel is assigned to the Measure channel of the current part. The assigned measure Kelvin channel number is displayed in the Net Editor as a Kelvin channel assigned to the measure channel net. For example, resistor R1 has a measure channel #6 assigned to its pin-2 (Net 5). If the Measure channel option is selected under the Kelvin column, the Net Editor will show that a Kelvin channel has been assigned to Net 5. If the Source & Measure option is selected, a Kelvin channel is assigned to both channels in the same manner. MeasureKelvin is designated as a collection in order to deal with multiple boards on one panel.
Public **ForcingValue** As Double - This is the voltage or current listed under the “Forcing Val” column in the Part Editor grid that is applied to the selected component.

Public **TripPoint** As Double - This double is the current limit for the ICAM voltage source, or the voltage limit for the current source as listed under the “TripPoint” column in the Part Editor Grid.

Public **TestTime** As Double - This double is the desired duration of the test. For certain tests which use a sine wave, this is the frequency of the applied sine wave.

Public **ExpectedValue** As Double - This is the normal value of the part being tested. It is usually imported from the project CAD files.

Public **PlusExpected** As Double - This value is the largest acceptable measured ActualValue. It is calculated using the ExpectedValue and the +TOL percentage.

Public **MinusExpected** As Double - This value is the smallest acceptable measured ActualValue. It is calculated using the ExpectedValue and the -TOL percentage.

Public **Comment** As String - This string represents the programmer’s comments about the test, except for a User test, where it references the name of a user-generated subroutine in the Basic Tab.

Public **Guard** As Collection - The Guard collection is a list of channels to be guarded as designated in the Guard Editor. In order to deal with multi-board panels, this is actually a “collection of collections”, analogous to a 2-dimensional array. The syntax by which a channel is referenced is `pd.Guard(pd.BoardNumber)(index)` in User Code.

Public **ActualValue** As Double - This is the actual measured value returned by the functional call. It is used in determining a Pass or Fail flag using the **PlusExpected** and **MinusExpected** values.

Public **Calibration** As Collection - This is a collection of capacitance calibration factors created for a capacitor test by selecting **Execute| Calibrate** from the **Parts-Debug** tab.
Public **UseCalibration** As Boolean - This boolean value is only active on capacitor tests. It determines if the fixture capacitance factors will be subtracted from the measured capacitance. This value is found under the Calibration column in the Part Editor.

Public **UseCapDischarge** As Boolean - When this boolean value is active, the capacitor test is performed and the ICAM voltage or current source is then programmed to 0 and left connected for a period of time equal to the specified test time.

Public **NumberOfSamples** As Integer - This is the number of times to execute the test. It is set to 1 by default. If this value is set to between 2 and 9, AutoGen returns the average of the measured values. If this value is 10 or greater, the "outlying" value is discarded and the rest of the measured values are averaged. The final result is returned in **ActualValue**.

Public **PassOp** As Integer - This property is used by the Visual Basic project executive to specify where execution should be directed if a test passes. It is not used by the programmer in AutoGen. Operation definitions for the Visual Basic project executive are found in the CYX.bas file in \Digalog\Include\.

Public **FailOp** As Integer - This property is used by the Visual Basic project executive to specify where execution should be directed if a test fails. It is not used by the programmer in AutoGen. Operation definitions for the Visual Basic project executive are found in the CYX.bas file in \Digalog\Include\.

Public **Location** As Integer - This property is a label in Visual Basic project executive code to which execution is directed by **PassOp** or **FailOp**. It is not used by the programmer in AutoGen.

Public **Condition** As Integer - This property is set to IPASS or IFAIL depending on the result of the test. Definitions of these conditions for the Visual Basic executive are found in the CYX.bas file in \Digalog\Include\.

Public **NumberOfBoards** As Integer - This is the number of boards on a multi-board panel as determined by the Panel Parameters frame under the Net Editor.
Public **BoardNumber** As Integer - The value of this property determines which board of a multi-board panel is being tested. It is located on the Panel Parameters frame under the Net Editor.

*** When referencing the MeasureChannel, SourceChannel, MeasureKelvin and SourceKelvin elements of the PartData object in a User test, use the syntax: `pd.MeasureChannel.Item(pd.BoardNumber)` which will properly reference the collection elements necessary to deal with multi-board panels. This syntax must be used even if you are dealing with a single-board fixture.

These properties are filled in before the BeforePartsTest() routine is called. The user can then use the properties to format the output in any way. The Actual property is the actual value of the part in its respective units. The ComponentType property is one of: RESISTANCE, CAPACITANCE, INDUCTANCE, DIO DEV, ZENER. This code is used in parttest.cls and agresult.bas.

**ShrtData.cls** - The ShortData class is defined in shrtdata.cls and contains the following properties:

Public **NetName** As String - This string is the name of the individual nets as listed under the Net Name column in the Net Editor.

Public **Channel** As Collection - This is the Matrix Relay channel assigned to a given net. Channel is designated as a collection in order to deal with multiple boards on one panel.***

Public **Resistance** As Double - This is the actual resistance measured by the ShortsMeas or OpenPinCheck functional calls. It is used by the program to determine shorts or opens and is not seen by the programmer.

Public **TestType** As Integer - This property determines which type of test is run. It is normally set to Shorts & Opens, but may be set to just Shorts, just Opens, or None. This column is found in the Net Editor.

Public **TestTime** As Double - This column in the Net Editor determines the duration of the test.
Public **ShortsThreshold** As Double - If the measured resistance between a net under test and any other net on the board is less than this value, a short is assumed to exist. The default value is set in the Nets - Shorts Test tab.

Public **OpensThreshold** As Double - If the measured resistance between a net under test and every other net on the board is greater than this value, an open is assumed to exist. The default value is set in the Nets - Shorts Test tab.

Public **Guard** As Collection - The Shorts test on a given net is performed by guarding (connecting to measurement analog ground) all nets on the board except the one to be tested, then applying a voltage to the net under test and measuring the resulting current. The Guard collection for a given net will consist of all nets on the board except for the net under test. Guard is designated as a collection in order to deal with multiple boards on a panel.

Public **PassOp** As Integer - This property is used by the Visual Basic project executive to specify where execution should be directed if a test passes. It is not used by the programmer in AutoGen. Operation definitions for the Visual Basic project executive are found in the CYX.bas file in \Digalog\Include\.

Public **FailOp** As Integer - This property is used by the Visual Basic project executive to specify where execution should be directed if a test fails. It is not used by the programmer in AutoGen. Operation definitions for the Visual Basic project executive are found in the CYX.bas file in \Digalog\Include\.

Public **Location** As Integer - This property is a label in Visual Basic project executive code to which execution is directed by PassOp or FailOp. It is not used by the programmer in AutoGen.

Public **Condition** As Integer - This property is set to IPASS or IFAIL depending on the result of the test. Definitions of these conditions for the Visual Basic executive are found in the CYX.bas file in \Digalog\Include\.

Public **NumberOfBoards** As Integer - This is the number of boards on a multi-board panel as determined by the Panel Parameters frame under the Net Editor.
Public **BoardNumber** As Integer - The value of this property determines which board of a multi-board panel is being tested. It is located on the Panel Parameters frame under the Net Editor.

*** When referencing the Channel and Guard collections in a User test, use the syntax: `sd.Channel.Item(sd.BoardNumber)` which will properly reference the collection elements necessary to deal with multi-board panels. This syntax must be used even if you are dealing with a single-board fixture.

These properties are filled in before the BeforeShortsTest() routine is called. The user can then use the properties to format the output in any way. The TestType property is NONE, SHORTS, SHORTS&OPENS, and OPENS which are defined in **icam32.bas**. This code is used in shrttest.cls and agresult.bas.

**AGData.cls** - The AutoGenData class is defined in agdata.cls and contains the following properties:

- Public Name As String
- Public PassOp As Integer
- Public FailOp As Integer
- Public Location As Integer
- Public Condition As Integer
- Public Test As New Collection
- Public NumberOfBoards As Integer
- Public BoardNumber As Integer

These properties are filled in before the BeforeBoard() routine is called. The BeforeBoard() routine is called before each board in a panel including a one panel board.

**cyx.bas** - Cyx.bas is the unchangeable code that interfaces to the code generated by AutoGen or Cyclops. This code contains the subroutines that interface to the CYX executive API. There are routines used to initialize the CYX executive API, execute the test, and an engineering notation format function. This code should not be changed because Digalog may modify it to include other functions.

The CYX executive API module contains the definition of InitializeCYX() and various constants: ICONT, IGOTO, ISTOP, IWAIT, IPASS and IFAIL. The first four are used for execution control and the last two are used to designate if a test is passed (IPASS) or failed (IFAIL).
**autogen.bas** - Autogen.bas is the unchangeable code that contains the constant definitions for AutoGen. Definitions for component type, shorts test type and test mode constants are all available.

### Interface to CYX Executive

The interface to the CYX executive API requires three parts: Initialization, Execution and Data Logging. Initialization is needed to setup the necessary classes with there respective data. Execution calls the subroutines located in the classes necessary to execute a set of tests. Data Logging is done for each test in the class. The interfaces to these classes are wrapped by Visual Basic subroutines located in the CYXAPI.bas module (described later).

#### CYX Executive API Initialization

In order to initialize all of the necessary data that the CYX executive API requires, some variables and subroutines must be declared, and called. The first is InitializeCYX(). This routine must be called at program startup or where the rest of the executive is initialized. This sets up the CYX executive API and initializes the classes that are generated from AutoGen and Cyclops. This function is declared in CyxAPI.bas (described later). Once this is done, all of the AutoGen and Cyclops objects are initialized and the CYX executive can be executed.

#### CYX Executive API Execution

Once the CYX executive API initializes all of the AutoGen and Cyclops classes, the CYX executive can be started by calling ExecuteCYX() like:

```plaintext
Dim pass As Integer
...
...  
pass = ExecuteCYX()
```

After ExecuteCYX() returns, pass contains one of two values: IPASS or IFAIL. These are defined in Cyx.bas. This notifies the calling routine if the CYX executive passed or failed. During the execution, there are hook subroutines that get called during execution. These routines are written by the user and are used for logging of data.
EXECUTIVE FLOW CHART

ExecuteCYX()
  Root
    BeforeCyxTests(cyxd As CyxData)
      Autogen
        BeforeAutoGenTests(agd As AutoGenData)
          BeforeBoard(agd As AutoGenData)
        ShortsTest
          BeforeAllShortsTest(agd As AutoGenData)
            Each Shorts Test
              BeforeShortTest(sd As ShortsData)
            ShortTest
              AfterShortTest(sd As ShortsData)
            EachO pensTest
              BeforeO penTest(sd As ShortsData)
              O penTest
              AfterO penTest(sd As ShortsData)
            AfterAllShortsTest(agd As AutoGenData)
          Parts Test
            BeforeAllParts(agd As AutoGenData)
            Each Parts Test
              BeforePartTest(pd As PartData)
              PartTest
              AfterPartTest(pd As PartData)
            AfterAllParts(agd As AutoGenData)
          User Test
            BeforeAllUser(agd As AutoGenData)
            Each User Test
              BeforeUserTest(pd As PartData)
              UserTest
              AfterUserTest(pd As PartData)
            AfterAllUser(agd As AutoGenData)
          AfterBoard(agd As AutoGenData)
        AfterAutoGenTests(agd As AutoGenData)
      Cyclops (If Present)
        AfterCyxTests(cyxd As CyxData)
CYX Executive API Data Logging

There are subroutines that need to be written before the CYX executive can be executed:

BeforeCyxTests(cyxd As CyxData)
AfterCyxTests(cyxd As CyxData)
BeforeAllTests(cd As CyxData)
AfterAllTests(cd As CyxData)
BeforeAllParts(cd As CyxData)
AfterAllParts(cd As CyxData)
BeforeAllShorts(cd As CyxData)
AfterAllShorts(cd As CyxData)
BeforePartsTest(pd As PartData)
AfterPartsTest(pd As PartData)
BeforeShortsTest(sd As ShortsData)
AfterShortsTest(sd As ShortsData)
BeforeOpenPinTest(sd As ShortsData)
AfterOpenPinTest(sd As ShortsData)

They should be located in a user specific .bas module (for example, agresult.bas) and included in the Visual Basic project. The PartData, ShortsData and CyclopsData parameters are class objects that get passed to the routines after each test executed. They contain all of the information necessary to log information about a specific test. These will be described later. There are various properties within PartData, ShortsData and CyclopsData that the user can use to format and determine results of each test.

The function BeforeAllShortsTest(cyxd As CyxData) gets called before any sub tests are executed. This allows the user to modify the execution flow. The cd.Location parameter is the first test to start executing. Before using this parameter, a knowledge of how many tests are in the current test object is necessary and can be obtained using cd.test.count().

The function BeforeShortTest(sd As ShortsData) sets conditions for the ShortTest function. Once the ShortTest function has been executed, the AfterShortTest(sd As ShortsData) function is called. The pass/fail data is determined, and the test number is incremented. The BeforeShortTest(sd As ShortsData) function is again called, and the next test in the sequence is executed. This cycle is repeated for every test in the sequence. User code may also be added to the before and after functions as required. The OpenPinTest
and PartsTest are handled in a similar manner.

The function AfterAllShortsTest(cyxd As CyxData) gets called after the sub test is executed. The PassOp and FailOp parameters are set from the sub test object. This is done so that the test can be interrupted if necessary. The Condition parameters is used to determine if the test passed (IPASS) or failed (IFAIL).

**fixture.bas**

This interface contains unchangeable code to initialize and use the vacuum controller. InitializeVacuum() initializes the vacuum to a known default state: all fixtures up. Fixture(side As Integer, position As Integer) allows the control of the fixture. “Side” can be DUT1 or DUT2 which are constants. “Position” can be DUTRaise or DUTLower which are also constants.

**Vbib32.bas**

This unchangeable code is the interface used by fixture.bas

**Niglobal32.bas**

This module contains the variable declarations, constant definitions, and type information that is recognized by the entire application.

**Icam32.bas**

This unchangeable code is the interface used by the AutoGen generated code.

**Directory Structure**

An example of the directory structure for the Sample project is shown to the right. AutoGen creates the Sample.hwc, Sample.pin, Sample.prt, and Sample.tst files in the AutoGen subdirectory of the Sample project. When the Generate Executive option from the AutoGen File menu is selected, the CYX executive code is generated in the file structure shown to the right.

1. The Agresult.bas and usertest.cls files are copied from the Digalog\Include\Cyx directory into the
Digalog\Projects\Sample\Autogen directory.

2. The Cyx.frm form from the Digalog\Include directory is copied and renamed Sample.frm in the Digalog\Projects\Sample\Autogen directory.

3. The Visual BASIC project file Sample.vbp is created in the Digalog\Projects\Sample\Autogen directory.

4. The Parttest.cls and Shrttest.cls class modules are created in the Digalog\projects\Sample\Autogen\Cyx directory.

5. The AutoGen.cls and Root.cls modules are created in the Digalog\projects\Sample\AutoGen directory.

6. The Autogen.bas, Cyx.bas, Cyxdata.cls, Fixture.bas, GPIB.bas, Icam.bas, Partdata.cls, Shrtdata.cls, and rtbex.cls modules from the Digalog\Include directory are linked to the Sample.vbp project file.

When the Sample.vbp file is opened from Visual BASIC, the files mentioned in steps 1 to 5 on the previous page appear in the project window as shown to the right. The standard CYX Executive (as shown on page 44) may be run directly without further modification. The form, however, may be modified to suit the user’s application. If so, the project may be saved with the desired modifications to the form. It will NOT be overwritten if the tests are modified under AutoGen. The only files that will be overwritten are the generated files under the Digalog\Projects\Sample\Autogen\Cyx directory.

The code in Agresult.bas may also be modified and saved within the project. Any additional user routines can be included in the appropriate spots within this file as indicated in the Executive Flow Chart shown on page 54.
Sample Functional Call

The following section contains the Digalog functional calls for the ICAM Executive program. Visual Basic declarations and parameters are shown for each call and follow the format of this page. The functional name is at the top. An explanation paragraph follows and then the declaration, syntax, and a list of the meanings of the parameters if any. The last section is an example of the functional call.

Visual Basic Declaration:
Public Sub Sample(ByVal Time As Long, ByVal Voltage As Double, ByVal Current As Double)

WHERE:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>1</td>
<td>1000</td>
<td>Time in seconds.</td>
</tr>
<tr>
<td>Voltage</td>
<td>0</td>
<td>300</td>
<td>Voltage in volts.</td>
</tr>
<tr>
<td>Current</td>
<td>0</td>
<td>300</td>
<td>Current in amps.</td>
</tr>
</tbody>
</table>

Call Sample(Time, Voltage, Current)
The AltSrcRMeas functional call is used to measure resistance by using an external source. When using this function, it is important to restrict the external source to 15 volts because if the DUT was shorted (a short circuit), the source voltage could compromise the ICAM current measurement amplifier.

**Visual Basic Declaration:**

```vbnet
Public Sub AltSrcRMeas(ByRef Resistance As Double, ByVal MeasureICh As Integer, ByVal ForcingV As Double, ByVal TextTime As Double, ByVal ExpRes As Double)
Call AltSrcRMeas(Resistance, MeasureICh, ForcingV, TextTime, ExpRes)
```

**WHERE:**

- **Resistance**
  - The measured resistance returned in ohms.

- **MeasureICh**
  - The current measurement channel connecting the DUT to the Matrix Relay channel.

- **ForcingV**
  - The voltage of the external source. Used for calculating the current measurement range. \( I_{range} = \frac{\text{ForcingV}}{\text{ExpRes}} \)

- **TextTime**
  - The duration of the test in Seconds.

- **ExpRes**
  - The value of the expected resistance used in conjunction with the ForcingV to calculate the current measurement range.
**EXAMPLE:**

Dim RetRes As Double

Call SMux(0, 1, 80) ............................... ‘Route the DA through SMux 1 of the VSource bus.

Call DA(0, 3) ...........................................................................................'Set the D/A voltage.

Call AltSrcRMeas(RetRes, 81, 3, 0.1, 300000000) ........................................... ‘Measure the resistance.

  Call DA(0, 0)
  Call SMux(0, -1, 80) .............................. ‘Reset D/A and reset the VSource bus.
BetaMeas

The BetaMeas functional call executes a beta measurement on a transistor. A voltage is applied to the transistor’s collector, a constant current is applied to the base, and the emitter current is measured. Since a transistor’s beta is defined as the collector current ($I_c$) divided by the base current ($I_b$) and the collector current is calculated as the emitter current ($I_e$) minus the base current, the transistor’s beta is calculated by \((I_e - I_b) / I_b\).

Before executing this functional call a voltage must be applied to the transistor’s collector. Typically the source would be one of the Analog Source Board’s DAs. This voltage source can be wired to one of the ICAM’s SMUX inputs. This voltage is then routed to the ICAM’s voltage source bus via the ‘SMux’ functional call which also connects the matrix relay channel connected to the transistor’s collector to the voltage source bus.

The ‘BetaMeas’ functional call is then used to take the measurement. The channel associated with the transistor’s base will be connected to the ICAM’s current source bus so the base current can be applied. The channel associated with the emitter will be connected to the ICAM’s measurement bus so the current out of the emitter can be measured. The given base current will be applied for the given test duration. At the end of the test duration the emitter current is measured and the transistor’s beta will be calculated. The value of the transistor’s beta is calculated by the equation: \((I_e - I_b) / I_b\).

Visual Basic Declaration:

```
Public Sub BetaMeas(ByRef RetBeta As Double, ByVal BaseChan As Integer, ByVal EmitterChan As Integer, ByVal BaseCurrent As Double, ByVal Limit As Double, ByVal TestTime As Double, ByVal ExpAmps As Double)
```

WHERE:

```
RetBeta = The measured Beta.
```
**BaseChan**

```plaintext
= 0 to last available Matrix Relay channel. The channel connected to the transistor’s base. This channel is also connected to the current source bus and is used to source the constant current into the base.
```

**EmitterChan**

```plaintext
= 0 to last available Matrix Relay channel. The channel connected to the transistor’s emitter. This channel is also connected to the measurement bus and is used to measure the emitter’s current.
```

**BaseCurrent**

```plaintext
= -0.050 to +0.050 Amps. The current applied to the base.
```

**Limit**

```plaintext
= 3.0 to 35.0 Volts. The voltage the current source’s output will be limited to.
```

**TestTime**

```plaintext
= 0 to 650 sec. The duration of the test (in seconds).
```

**ExpAmps**

```plaintext
= 0.0 to 0.100 Amps. The expected current measured at the emitter.
```

**EXAMPLE:**

Dim BetaVal As Double

Call IcamSamples(1) ............................................... ‘Set the number of samples to read to one.

Call IcamDelay(0.9) ................................. ‘Set the percentage of test time to allow for settling.

Call DA(0, 5.0) ............................................................................. ‘Apply the collector voltage.

Call SMux(0, 1, 65) ................... ‘Setup the ICAM’s SMux to route the DA voltage from the DA through SMux bus #1 (V source bus) channel #0 to the matrix relay channel connected to the transistor’s collector (channel #65)

Call BetaMeas(BetaVal, 56, 62, 250e-6, 3.0, 0.001, 0.100) ..................... ‘Connect matrix relay channel 56 to the current source bus and the transistor’s base and channel 62 to the measurement bus and to the emitter, source 250 microAmps into the base, limit the voltage at the current source’s output to 3 volts, the test duration will be 1 milliSec, and the expected current measured at the emitter should be less than 100 milliAmps. The calculated Beta will be returned in the parameter “BetaVal”.

```
The CDischarge functional call is used to dissipate a charge present across a capacitor to less than a specified voltage level. The function relies on an external discharge resistor (in the fixture) to shunt and limit the current generated by the discharge. The resistor is connected between Analog ground and an ICAM SMux input on buses 0, 1, or 2.

The functional call applies the external resistor across the capacitor and begins monitoring the capacitor voltage. When it is less than the predetermined value, the load is removed and the function exits. If the timeout elapses before the value is reached, the function returns immediately with an error indicating this condition.

**Note:** All switching circuitry is limited to 500mA of current and 200V. Please select the external load accordingly.

**Visual Basic Declaration:**

```vbnet
Public Sub CDIscharge(ByVal HiChan As Integer, ByVal LoChan As Integer, ByVal SMuxBus As Integer, ByVal SMuxInput As Integer, ByVal Limit As Double, ByVal TimeOut As Double)
Call CDIscharge(HiChan, LoChan, SMuxBus, SMuxInput, Limit, TimeOut)
WHERE:

HiChan = 0 to last available Matrix Relay channel - The channel used for the high side of the capacitor (reference only).

LoChan = 0 to last available Matrix Relay channel - The channel used for the low side of the capacitor (reference only).

SMuxBus = 0 to 2. The desired SMux bus on the ICAM board. Bus 3 is reserved for this function and should not be used.

SMuxInput = 0 to 3. The SMux channel the bus is connected to.
```
**Limit**

= The voltage limit in volts. This is the voltage level to discharge the capacitor to (max).

**TimeOut**

= Time for the capacitor to fully discharge in seconds. This parameter should include additional time (over the calculated time) to provide a margin of safety.

Large capacitors, especially electrolytic capacitors, recover somewhat after discharge to regain a portion of their former voltage. This is due to the relatively large built-in series resistance, and could cause the capacitor to regain a significant portion of its former charge.

**EXAMPLE:**

Call CDischarge(66, 67, 2, 0, 0.1, 3.5) ...................... ‘Discharge a 35 Volt 5000uF capacitor (maximum) across channels 66 and 67, through a 100 ohm resistor connected to SMUX bus 2, input 0. Use a 100 mV Limit and a 3.5 second (2.9s calculated) TimeOut.
CMeas

The CMeas functional call measures the capacitance between the source channel and the measure channel. A voltage ramp is applied to the source channel for the duration of the test time, then the current is measured at the measure channel. The returned capacitance is calculated by dividing the measured current by the applied voltage ramp. An expected capacitance is required to allow the measurement circuitry to adjust to the appropriate range.

Note: If the “Expected” parameter is equal to or greater than 47uF, a two ohm resistor is placed in series with the source channel to limit the inrush current.

Visual Basic Declaration:

Public Sub CMeas(ByRef Capacitance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal SrcVoltage As Double, ByVal Limit As Double, ByVal TestTime As Double, ByVal Expected As Double)

WHERE:

Capacitance = The measured capacitance returned in Farads.

Source = 0 to last available Matrix Relay channel - The channel used as a voltage source channel.

Measure = 0 to last available Matrix Relay channel - The channel used as a measurement channel.

SrcVoltage = The voltage ramp (in Volts/Sec) applied to the channel - The peak voltage must not exceed +2.5V. The peak voltage is calculated as the ramp voltage multiplied by the test time.
**Limit**

- **Limit** = 0 to 1.0 - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 Amps, and anything else limits the circuit to 1.0 Amps.

**TestTime**

- **TestTime** = 0 to 650 - The duration of the test in Seconds.

**Expected**

- **Expected** = The expected capacitance, in Farads.

**EXAMPLE:**

```vbnet
dim CRet as Double
Call CMeas(CRet, 43, 193, 1.0, 1.0, 0.100, 0.000001) 'Measure capacitance between channels 43 and 193. Use a ramp function to 1 V, up to 1 A for 100 mS. Expect a capacitance of about 1 uF.
```
The CofRCMeas functional call applies a voltage ramp to the DUT and the resulting current is measured. It assumes that the value being passed in the “ExpRes” parameter is valid. The capacitance is calculated as:

\[ C = (I_{TOT} - I_{RES}) \times \frac{dv}{dt} \]

Where:
- \( I_{TOT} \) = measured current
- \( I_{RES} \) = calculated current due to resister
- \( \frac{dv}{dt} \) = source ramp

When using CofRCMeas with reference to the following example, a load resistor is wired between SMux bus 1, Input 2 (Left pin of row 17 of the ICAM board) to analog ground (Right pin of row 11 of the ICAM board). The load resistor should be selected in order to discharge the largest capacitor on the product as fast as possible while keeping the discharge current below one-half ampere.

Visual Basic Declaration:

```vbnet
Public Sub CofRCMeas(ByRef Capacitance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal ForcingV As Double, ByVal Limit As Double, ByVal ExpCap As Double, ByVal ExpRes As Double)
  Call CofRCMeas(Capacitance, Source, Measure, ForcingV, Limit, ExpCap, ExpRes)
End Sub
```

WHERE:

- **Capacitance**
  - The measured capacitance returned in Farads.

- **Source**
  - 0 to last available Matrix Relay channel - The channel used as a voltage source channel.

- **Measure**
  - 0 to last available Matrix Relay channel - The channel used as a measurement channel.
ForcingV  
\[ \text{ForcingV} = \text{-2273V to 2273V - The peak voltage must not exceed +2.5V. The peak voltage is calculated as the ramp voltage multiplied by the test time plus approximately a 10% overhead. Because the test time for this test is always .001, the peak voltage can be calculated as: } \text{[Vp = ForcingV * .001 * 1.1]} \]

Limit  
\[ \text{Limit} = 0 \text{ to 1.0 - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 Amps, and anything else limits the source to 1.0 Amp.} \]

ExpCap  
\[ \text{ExpCap} = \text{Expected capacitance between source and measure channels.} \]

ExpRes  
\[ \text{ExpRes} = \text{Expected resistance between source and measure channels.} \]

**EXAMPLE:**

Dim Capacitance As Double

Call CofRCMeas(Capacitance, 171, 172, 2200, 1, .000000047, 3300)

........................................................................'Measure the capacitance of the RC network
The CofRCMeas2 functional call applies a sinusoidal voltage to one side of the RC DUT and the other side is “Guarded”. This function assumes that the value of parallel resistance being passed in the “ExpRes” parameter is valid. The current through the DUT is measured and the capacitance is calculated as follows:

\[
C = \frac{R \times I_{\text{RMS}} \times V_{\text{RMS}}}{R \times V_{\text{RMS}} \times 2 \times \pi \times f}
\]

Where:
- \( R \) = ExpRes
- \( I_{\text{RMS}} \) = Measured current
- \( V_{\text{RMS}} \) = Source voltage
- \( f \) = Source frequency

Visual Basic Declaration:

```
Public Sub CofRCMeas2(ByRef Capacitance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal PeakV As Double, ByVal Limit As Double, ByVal Frequency As Double, ByVal ExpCap As Double, ByVal ExpRes As Double)
Call CofRCMeas2(Capacitance, Source, Measure, PeakV, Limit, Frequency, ExpCap, ExpRes)
```

WHERE:

- **Capacitance**
  = The measured capacitance returned in Farads.

- **Source**
  = 0 to last available Matrix Relay channel - The channel used as a voltage source channel.

- **Measure**
  = 0 to last available Matrix Relay channel - The channel used as a measurement channel.

- **PeakV**
  = -2.5V to 2.5V - The peak amplitude of the applied sinusoidal source.

- **Limit**
  = 0 to 1.0 - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 Amps, and anything else limits the source to 1.0 Amps.
**Functional Calls**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frequency</strong></td>
<td>0 to 150,000. Frequency of the applied sinusoidal source.</td>
</tr>
<tr>
<td><strong>ExpCap</strong></td>
<td>Expected capacitance between the source and measure channels.</td>
</tr>
<tr>
<td><strong>ExpRes</strong></td>
<td>Expected resistance between the source and measure channels.</td>
</tr>
</tbody>
</table>

**EXAMPLE:**

```vbnet
Dim Capacitance As Double
Call CofRCMeas(Capacitance, 171, 172, 0.4, 0.3, 123000, .000000047, 3300)
.............................................................................. 'Measure the capacitance of the RC network.
```
**CofRCMeas3**

The CofRCMeas3 functional call applies a sinusoidal voltage to one side of the RC DUT. This voltage is also applied to one of the crossover detectors on the ICAM board. The current waveform through the DUT is applied to the other crossover detector on the ICAM board, and the difference in phase angle is measured. The capacitance of the DUT is calculated as follows:

$\phi$ = Difference in Phases

$$X_c = \frac{R}{\tan(\phi)}$$

$$C = \frac{1}{2\pi f X_c}$$

**Visual Basic Declaration:**

```vbnet
Public Sub CofRCMeas3(ByRef Capacitance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal PeakV As Double, ByVal Limit As Double, ByVal Frequency As Double, ByVal ExpCap As Double, ByVal ExpRes As Double)
Call CofRCMeas3(Capacitance, Source, Measure, PeakV, Limit, Frequency, ExpCap, ExpRes)
```

**WHERE:**

- **Capacitance** = The measured capacitance returned in Farads.
- **Source** = 0 to last available Matrix Relay channel - The channel used as a voltage source channel.
- **Measure** = 0 to last available Matrix Relay channel - The channel used as a measurement channel.
- **PeakV** = -2.5V to 2.5V - The peak amplitude of the applied sinusoidal source.
**Limit**

\[ Limit = 0 \text{ to } 1.0 \text{ - Current limit of the voltage source. Values less than or equal to } 0.2 \text{ fix the limit at } 0.2 \text{ Amps, and anything else limits the source to } 1.0 \text{ Amps.} \]

**Frequency**

\[ Frequency = 0 \text{ to } 10,000 \text{. Frequency of the applied sinusoidal source.} \]

**ExpCap**

\[ ExpCap = \text{Expected capacitance between the source and measure channels.} \]

**ExpRes**

\[ ExpRes = \text{Expected resistance between the source and measure channels.} \]

**EXAMPLE:**

```vbnet
Dim Capacitance As Double
Call CoRCM eas3(Capacitance, 171, 172, 0.4, 0.3, 10000, .000000047, 3300)
```

.......................... 'Measure the capacitance of the RC network.
The CPol functional call sources a voltage into the positive lead of a capacitor and measures the leakage current present on the negative lead of the capacitor. If the capacitor is incorrectly installed, the leakage current through the capacitor will be significantly higher than the leakage current found to be nominal during test development. The test time may be extended to provide for charging large capacitors, causing a greater difference between the normal leakage current and the reverse polarity leakage current.

The expected current should be selected such that the maximum resolution will be achieved for reverse polarity leakage current without pushing the measurement to a rail. If the measurement is at the rail, the returned current will be identical for successive executions.

Test parameters must be manipulated to obtain optimum reading separation and ensuring steady-state measurements. The source voltage must be limited such that leakage current greater than 100 mA is not achieved. That is the upper limit of the current measurement circuit's range. The range must be selected using the ExpCurr parameter such that correctly polarized and incorrectly polarized leakage currents may be accurately measured.

The TestTime must be selected such that the capacitor is charged to the source voltage, and a steady-state condition is reached. The capacitor charge-up time is partly dependent on the feedback resistor of the selected measurement amplifier range, so as the range is adjusted (with ExpCurr), so might the TestTime need to be.

The CDischarge function should be run after the test to remove any charge which may still be present on the capacitor.

**Note:** The example code requires the use of an external voltage source connected to a SMux Bus Input (Bus 0 or 2). The example code also uses a 100 ohm fixture resistor to discharge the capacitor when testing has been completed. The resistor is connected to an analog ground pin (right pin of row 12 of the ICAM slot) in the fixture as well as SMux input 2 of bus number 2 (left pin of row 18 of the ICAM slot).
Visual Basic Declaration:
Public Sub CPol(ByRef LeakageCurrent As Double, ByVal Plus As Integer, ByVal Minus As Integer, ByVal SMuxBus As Integer, ByVal SMuxInput As Integer, ByVal TestTime As Double, ExpCurr As Double)

Call CPol(LeakageCurrent, Plus, Minus, SMuxBus, SMuxInput, TestTime, ExpCurr)

WHERE:

LeakageCurrent
   = The measured current returned in Amps.

Plus
   = 0 to last available voltage Matrix Relay channel - The channel connected to the positive side of the capacitor.

Minus
   = 0 to last available Matrix Relay channel - The channel connected to the negative side of the capacitor.

SMuxBus
   = 0 or 2. The desired SMux bus on the ICAM board. For an external voltage source, Bus 0 is reserved. Bus 3 is reserved when guarding during this test.

SMuxInput
   = 0 to 3. The SMux channel the bus is connected to.

TestTime
   = 0 to 650 sec - The duration of the test (in Seconds).

ExpCurr
   = The expected leakage current in Amps.

EXAMPLE:
Dim RetC As Double
Call DA(0, 6)  'Generate 6V external source for test.
Call CPol(RetC, 112, 106, 0, 0, 0.1, 0.1)  'Apply source to channel 112 and measure leakage at channel 106. Use SMux Bus 0 and SMux channel 0.
Call DA(0, 0)  'Turn off source.
Call CDischarge(112, 106, 2, 2, 0.1, 1)  'Discharge DUT.
### DVM_meas

The DVM_eas functional call measures the voltage drop across a forward-biased p-n junction. A constant current is applied to the source channel (anode) for the duration of the test time, and the cathode should be connected to the Guard Bus. The returned voltage is measured at the source, and will equal the forward voltage drop of the p-n junction.

#### Visual Basic Declaration:

```vbnet
Public Sub DVMeas(ByRef Voltage As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal SrcCurrent As Double, ByVal Limit As Double, ByVal TestTime As Double, ByVal Expected As Double)
Call DVMeas(Voltage, Source, Measure, SrcCurrent, Limit, TestTime, Expected)
```

#### Where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Voltage</strong></td>
<td>The measured forward voltage returned in Volts.</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>0 to last available voltage Matrix Relay channel - The channel used as a current source channel.</td>
</tr>
<tr>
<td><strong>Measure</strong></td>
<td>0 to last available Matrix Relay channel - The channel connected to the Guard Bus.</td>
</tr>
<tr>
<td><strong>SrcCurrent</strong></td>
<td>0 to 50mA - Current applied to the source channel (in Amps).</td>
</tr>
<tr>
<td><strong>Limit</strong></td>
<td>3.0V to 35.0V - The voltage limit on the current source (in Volts).</td>
</tr>
<tr>
<td><strong>TestTime</strong></td>
<td>0 to 650 sec - The duration of the test (in Seconds).</td>
</tr>
<tr>
<td><strong>Expected</strong></td>
<td>The expected return voltage (in Volts).</td>
</tr>
</tbody>
</table>
**EXAMPLE:**

Dim VRet as Double

Call DVM_meas(VRet, 43, 193, .01, 5.0, 0.01, 0.7) ................. 'Measure the forward voltage drop
................................................................................ of a p-n junction from channel 43 to 193.
............................................................................... Use a constant current function of 10 mA, up to 5V for 10 mS.
............................................................................... Expect a drop of about 0.7 V. (Samples may be taken and averaged.)
DZMeas

The DZMeas functional call measures the voltage drop across a Zener diode. A constant current is applied to the source channel (cathode) for the duration of the test time, and the anode should be connected to the Guard Bus. The returned voltage is measured at the source, and should equal the reverse breakdown voltage of the p-n junction.

Visual Basic Declaration:

```
Public Sub DZMeas(ByRef Voltage As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal SrcCurrent As Double, ByVal Limit As Double, ByVal TestTime As Double, ByVal ExpVolt As Double)
```

WHERE:

- **Voltage**
  
  The measured breakdown voltage returned in Volts.

- **Source**
  
  The channel used as a current source channel.

- **Measure**
  
  The channel connected to the Guard Bus.

- **SrcCurrent**
  
  Current applied to the source channel (in Amps).

- **Limit**
  
  Voltage limit on the current source (in Volts).

- **TestTime**
  
  The duration of the test (in Seconds).

- **ExpVolt**
  
  The expected return voltage (in Volts).
**EXAMPLE:**

Dim VRet as Double

Call DZMeas(VRet, 43, 193, .01, 7.0, 0.01, 5.2) ........... ‘Measure the reverse-bias voltage drop of a p-n junction from channel 43 to 193. Use a constant current function of 10 mA, up to 7 V for 10 mS. Expect a drop of about 5.2 V. (Samples may be taken and averaged.)
The FixID functional call retrieves the identification code of the test fixture installed on the Patchboard receiver. Sixteen Patchboard pins are provided by the ICAM board to the fixture. Any bits which are connected to the Patchboard digital ground pin in the fixture are read as a 0 bit by the receiver, and the remaining bits will be pulled high and read as a 1.

Visual Basic Declaration:
Public Sub FixID(ByRef Code As Integer)
Call FixID(Code)
WHERE:
Code = 16-Bit Fixture code read by the functional call.

EXAMPLE:
Dim Code As Integer
Call FixID(Code) ............................................................... ‘Retrieve fixture identification code.
FixStat

The FixStat functional call checks the status of the vacuum wells in the fixture (two maximum). If the fixture is built with a contact switch in each vacuum well, the status of these switches may be read to determine if the UUT is engaged for testing.

Visual Basic Declaration:
Public Sub FixStat(ByVal Stat1 as Integer, ByVal Stat2 as Integer)

WHERE:

\[
\begin{align*}
\text{Stat1} &= 0 \quad \text{UUT #1 is raised} \\
&= 1 \quad \text{UUT #1 is lowered.}
\end{align*}
\]

\[
\begin{align*}
\text{Stat2} &= 0 \quad \text{UUT #2 is raised} \\
&= 1 \quad \text{UUT #2 is lowered.}
\end{align*}
\]

EXAMPLE:
Dim Stat1 As Integer
Dim Stat2 As Integer
Call FixStat(Stat1, Stat2) 'Retrieve fixture vacuum well status.
The Guard functional call sets a single Matrix Relay channel to the Guard Bus. NOTE: The Guard functional call is a “deferred” call. The Guard function will not take place until an “immediate” call such as a measurement call is invoked.

Visual Basic Declaration:
Public Sub Guard(ByVal Channel As Integer)
Call Guard(Channel)
WHERE:
Channel = 0 to last available Matrix Relay channel.

EXAMPLE:
Call Guard(83) .......................................................... Guard channel 83.
GuardBd

The GuardBd functional call connects all of the Matrix Relay channels on a given board to the Guard Bus.

**Note:** The GuardBd functional call is a “deferred” call. The GuardBd function will not take place until an “immediate” call such as a measurement call is invoked.

**Visual Basic Declaration:**
```
Public Sub GuardBd(ByVal MrlyBoardNum As Integer)
Call GuardBd(MrlyBoardNum)
```

**WHERE:**

- **MrlyBoardNum**
  - **= 0** Mrly channels 0 to 63.
  - **= 1** Mrly channels 64 to 127, etc. to 15.

Board number whose channels are to be connected to the Guard Bus.

**EXAMPLE:**
```
Dim MrlyBoardNum As Integer

For MrlyBoardNum = 0 To 1
    Call GuardBd(MrlyBoardNum)
Next MrlyBoardNum
```

‘Connect Matrix Relay channels 0 to 127 to the Guard Bus.'
**HFMux**

The HFMux functional call routes one of eight high-frequency inputs to a measurement system input. The HF inputs are arranged in pairs, each of which goes to one of four range select circuits. Each range select circuit may be used by the ICAM board’s Voltage Measurement System or by the Amplitude Measurement System (as SigA, Sig1, or Sig2 signals). Only one channel in each pair may be used at a time.

(The input path is disconnected if HFMux is followed by a later MMux or HFMux call to the same bus/range circuit or a component measurement is taken).

**Visual Basic Declaration:**

```
Public Sub HFMux(ByVal Channel As Integer, ByVal Sig As Integer, Range As Integer)
Call HFMux(Channel, Sig, Range)
```

**WHERE:**

**Channel**

- `-1` Disconnect all high frequency inputs
- `0` or 1, connect channel 0 or 1 to voltage range circuit 0.
- `2` or 3, connect channel 2 or 3 to voltage range circuit 1.
- `4` or 5, connect channel 4 or 5 to voltage range circuit 2.
- `6` or 7, connect channel 6 or 7 to voltage range circuit 3.

**Sig**

- `-1` Use ICAM Voltage Measurement Circuitry
- `0` AMS SigA
- `1` AMS Sig1
- `2` AMS Sig2.

The AMS Signals may also be used to generate TrigA, Trig1, and Trig2.

**Range**

- `0.1` 0.2 volt full scale
- `1.0` 2 volts full scale
- `10` 20 volts full scale
- `100` 200 volts full scale.
EXAMPLE:

Call HFMux(-1, 0, 0) ........................................................................... ‘Disconnect all HFMux inputs.

Call HFMux(4, -1, 10.0) ........................................ ‘HFMux input 4 to ICAM Voltage Measurement System ................................................................. using 20 volt range (via range circuit 2).
ICAM Clear

The ICAM Clear functional call resets and initializes the ICAM board and all of the Matrix Relay boards. The initialization consists of setting up the ICAM’s four MDA buses for their intended use and disconnecting all of the Matrix Relay channels from the Source, Measure, and Guard buses.

**Note:** Since this is the only functional call that searches for the Matrix Relay boards, it must be called at the start of any program that uses the ICAM board.

**Visual Basic Declaration:**
```
Public Sub IcamClear()
```

**Call ICAM Clear**

**EXAMPLE:**

```
Call ICAM Clear() .......................................................... 'Resets and initializes the ICAM board,
............................................................................ and disconnects all of the Source, Measure,
........................................................................................... and Guard channels from their respective buses.
```
ICAMDelay

The ICAMDelay functional call sets the delay that occurs after the generated source function is applied and before any samples or readings are taken. The delay parameter is a fraction of the total test time given as a parameter in a measurement call. An explanation of the timing of an ICAM measurement is included at the end of this section.

This function has its primary use with the component measurement functions. It is used to wait for the measurement node to settle before taking any samples. The main reason the measurement node would need to settle is because a capacitor on that node is causing oscillations when the circuit is first energized.

**Visual Basic Declaration:**

```vbscript
Public Sub IcamDelay(ByVal Delay As Double)
```

**WHERE:**

```
Delay = 0.0 to 0.95. The fraction of total test time to hold off readings.
```

**EXAMPLE:**

Call ICAMDelay(0.0) ............................................................ 'No sample delay (begin sampling as soon as the sourcing function is applied).

Call ICAMDelay(0.50) .................................. 'Set the sample delay to 50% of overall test time.
ICAMGetSamples

The ICAM GetSamples functional call retrieves the samples from the A/D converter on the ICAM board after a measurement functional call. ICAM Samples must be called before this function is executed to set the proper number of samples to store.

The measurement functional call determines the units on the values in the returned data array. If the test involved current readings (i.e. RM eas, CM eas), then the returned values will be in Amps; if the test involved voltage readings (i.e. DVM eas, DZM eas), then the returned values will be in Volts.

**Visual Basic Declaration:**

```vba
Public Sub IcamGetSamples(ByRef SampleArray() As Double, ByVal NSamples As Long)
Call ICAMGetSamples(SampleArray, NSamples)
```

**WHERE:**

- **SampleArray**
  - The array to store sample information. Must have at least NSamples elements.

- **NSamples**
  - 1 to 32768. The size of the array.

**EXAMPLE:**

```vba
Dim Samples(99) As Double
Call ICAMGetSamples(Samples,100).............. 'Retrieve 100 readings from the sample memory.
```
**ICAMOCData**

The ICAMOCData functional call sets the state of one of the four open-collector drivers on the ICAM board. These drivers require external pull-up connections for proper operation. By default, the reset state of the drivers is high (off).

**Visual Basic Declaration:**
```
Public Sub ICAMOCData(ByVal Channel As Integer, ByVal State As Integer)
Call ICAMOCData(Channel, State)
```

### WHERE:

**Channel**

- = -1  Reset all drivers to the high state (off).
- = 0  to 3 indicates an Open-collector channel.

**State**

- = 0  Indicates Driver output goes high (off state).
- = 1  Indicates Driver output goes low (on state).

### EXAMPLE:

```
Call ICAMOCData(-1, 0) 'Turn off all drivers.
Call ICAMOCData(2, 1) 'Turn on driver 2.
Call ICAMOCData(3, 0) 'Turn off driver 3.
```
The IcamRangeErrorEnable functional call enables or disables the ICAM A/D Out Of Range error generation. With the error generation enabled, a 101:013 Icam Measurement Out of Range error will be returned by any ICAM measurement call if the ICAM A/D converter rails during the measurement.

Visual Basic Declaration:

Public Sub IcamRangeErrorEnable(ByVal State As Integer)
Call IcamRangeErrorEnable(State)
WHERE:

State

= ICAM A/D Out Of Range error generation enable.
= 0 Disable Out Of Range error generation, does not return 101:013 error if A/D rails. (default)
= 1 Enable Out Of Range error generation, returns 101:013 error if A/D rails.

EXAMPLE:

Call IcamRangeErrorEnable(1) ................. Enable ICAM A/D Out Of Range error generation.
ICAMSamples

The ICAM Samples functional call sets the number of samples to take during the execution of a measurement functional call. All samples will be taken after the ICAM Delay specified and before the end of the test time given by the measurement call. The samples will be averaged to produce the value returned by the measurement function, and the data may be downloaded with the ICAM GetSamples function for graphing or analysis purposes.

Visual Basic Declaration:
Public Sub IcamSamples(ByVal NSamples As Long)

WHERE:

NSamples = 1 to 32768. The number of samples to take.

EXAMPLE:
Call ICAM Samples(1) ....................................................... ‘Take a single reading, no averaging.
Call ICAM Samples(100) ............................’Take 100 readings and average for the return value.
IcamSampleStatus

The IcamSampleStatus functional call returns the ICAM A/D status flag for the last set of A/D samples. The status flag is set if any of the A/D samples are out of range.

Visual Basic Declaration:
Public Sub IcamSampleStatus(ByVal Status As Integer)

WHERE:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 0</td>
<td>All samples within range of A/D.</td>
</tr>
<tr>
<td>= 1</td>
<td>A/D railed for one or more samples.</td>
</tr>
</tbody>
</table>

EXAMPLE:

```
Dim Status as Integer

Call IcamSampleStatus(Status) 'Get ICAM A/D status.
```
The IMeas functional call returns the current measurement on the ICAM current measurement bus. As a mid-level function export, it enables the user specific capabilities in the measurement of current.

**Visual Basic Declaration:**

```visualbasic
Public Sub IMeas(ByVal Amperage As Double, ByVal ExpCurrent As Double, ByVal TestTime As Double)
Call IMeas(Amperage, ExpCurrent, TestTime)
```

**WHERE:**

- **Amperage**
  - Measured return amperage in Amps.

- **ExpCurr**
  - The expected current in Amps. This value sets the appropriate measurement range.

- **TestTime**
  - 0 to 650 - The duration of the test in seconds.

**EXAMPLE:**

```vbnet
Dim RetCurr As Double
Call IMeas(RetCurr, 0.1, 0.001) 'Measure the current on the current measurement bus at 100 mA. Set TestTime at 1 mS
```
The \texttt{LMeas} functional call measures the inductance between the source channel and the measure channel. A sinusoidal voltage is applied to the source channel with the measurement channel grounded. The source current is measured to determine the impedance of the component at the given frequency. The inductance may then be calculated from the impedance and source frequency.

\textbf{Note:} This function calculates the number of required samples it needs to take. It will ignore any value entered with a prior ICAMSamples functional call.

\textbf{Visual Basic Declaration:}
\begin{verbatim}
Public Sub LMeas(ByRef Inductance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal SrcVoltage As Double, ByVal PeakI As Double, ByVal Frequency As Double, ByVal Expected As Double)
Call LMeas(Inductance, Source, Measure, SrcVoltage, PeakI, Frequency, Expected)
\end{verbatim}

\textbf{WHERE:}

\begin{align*}
\textbf{Inductance} & = \text{The measured inductance returned in Henrys.} \\
\textbf{Source} & = 0 \text{ to last available Matrix Relay channel - The channel used as a voltage source channel.} \\
\textbf{Measure} & = 0 \text{ to last available Matrix Relay channel - The channel used as a ground channel.} \\
\textbf{SrcVoltage} & = -2.5V \text{ to } 2.5V \text{ - Peak voltage of sinewave applied to the source channel in volts.} \\
\textbf{PeakI} & = \text{Saturation current of the inductor.} \\
\textbf{Frequency} & = 5KHz \text{ to } 200KHz \text{ - The frequency of the voltage sinewave source.}
\end{align*}
**Expected**

= The expected inductance (in Henrys).

**EXAMPLE:**

```
Dim LRet As Double

Call LMeas(LRet, 43, 193, 1.0, 1.0, 8000, 0.001) 'Measure inductance between channels 43 and 193. Use a voltage sinewave function at 1V peak at 8KHz, up to 1A peak. Expect an inductance of about 1mH. (Samples are handled internally.)
```
MKelvin

The MKelvin functional call enables or disables the measurement Kelvin capability of the ICAM board and selects the Matrix Relay channel to be used. If the channel number specified is positive, the ICAM board’s measurement Kelvin is enabled and routed to that Matrix Relay channel. If the channel parameter is negative, the ICAM board’s Kelvin measurement capability is disabled and the Matrix Relay channel used for measurement Kelvin is reset (if previously set).

Visual Basic Declaration

```
Public Sub MKelvin(ByVal Channel As Integer)
Call MKelvin(Channel)
WHERE:
```

<table>
<thead>
<tr>
<th>Channel</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>= &lt; 0</td>
<td>Disconnects a previously set MKelvin channel and resets the measurement Kelvin circuit.</td>
</tr>
<tr>
<td>= 0</td>
<td>to the last available channel. The channel to connect to the MKelvin bus (MRly bus 2).</td>
</tr>
</tbody>
</table>

Example:

```
Call MKelvin(-1) ........................................... 'Disable the ICAM measurement Kelvin capability and disconnect the Matrix Relay channel previously connected as the source Kelvin channel. 
Call MKelvin(520) .............................. 'Connect MRly channel 520 to Measurement Kelvin bus and enable the ICAM measurement Kelvin capability. 
```
MMux

The MMux functional call routes a Matrix Relay channel to a measurement system input through one of four ICAM buses, each with a range select circuit. The output of any range select circuit may be used by the ICAM board’s Voltage Measurement system or by the Amplitude Measurement System (as SigA, Sig1, or Sig2).

(The signal path is disconnected if MMux is followed by a later SMux or HFMux call to the same bus/range circuit or a component measurement is taken.)

Visual Basic Declaration:
Public Sub MMux(ByVal Channel As Integer, ByVal Bus As Integer, ByVal Sig As Integer, ByVal Range As Double)

WHERE:

Channel
= -1 Resets the ICAM bus specified below.
= 0 to the last available Matrix Relay channel. Selects the Matrix Relay channel.

Bus
= -1 Reset all buses (to default MDA signa);
= 0 to 3 Bus used to connect input signal to channel output.

Sig
= -1 Use ICAM Voltage Measurement Circuitry.
= 0 AMS SigA
= 1 AMS Sig
= 2 AMS Sig2.

The AMS Signals may also be used to generate TrigA, Trig1, and Trig2.

Range
= 0.1 0.2 volt full scale
= 1.0 2 volts full scale
= 10 20 volts full scale
= 100 200 volts full scale.
EXAMPLE:

Call MMux(-1, 1, 0, 0) ............................................. 'Reset ICAM bus 1 to default MDA signal.
Call MMux(-1, -1, 0, 0) ......................................................................... 'Reset all ICAM buses.
Call MMux(37, 2, -1, 10.0) ....................... 'Channel 37 to ICAM Voltage Measurement System ............................................................... (using bus 2 and 20 volt range).
Call MMux(121, 0, 1, 1.0) .............. 'Channel 121 to AMS Sig 1 (using bus 0 and 2 volt range).
OpenPinCheck

The OpenPinCheck functional call executes an opens test on the given channel. This test is used to determine if a fixture pogo pin is in fact making contact with the UUT. Typically all of the channels will be connected to the voltage source bus by using the VSrcBd functional call prior to executing any opens test. OpenPinCheck will disconnect the given channel from the voltage source bus and connect it to the measurement bus. The given constant voltage will be applied to all of the nodes connected to the voltage source bus. Then the current is measured at the measurement node. This current should be equal to the source voltage divided by the parallel resistance connected to the measurement node. The calculated parallel resistance is returned to the user. It should be noted that before each opens test, the channel previously connected to the measurement bus is disconnected from that bus and reconnected to the voltage source bus.

After all of the nodes have been checked for opens, the UnVSrcBd functional call should be used to disconnect all of the channels from the voltage source bus before continuing.

Visual Basic Declaration:

```vbnet
Public Sub OpenPinCheck(ByRef RetRes As Double, ByVal MeasChan As Integer, ByVal SrcVoltage As Double, ByVal Limit As Double, ByVal TestTime As Double, ByVal ExpRes As Double)
Call OpenPinCheck(RetRes, MeasChan, SrcVoltage, Limit, TestTime, ExpRes)
WHERE:

RetRes = The measured resistance returned in Ohms.
MeasChan = 0 to last available Matrix Relay channel - The channel used as the measurement node.
SrcVoltage = -2.5 to +2.5 - The voltage applied to the Voltage Source Bus.
```
Limit
= 0.0 to 1.0 - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 amps. Anything else sets the limit to 1.0 amp.

TestTime
= 0 to 650 - The duration of the test in seconds. The measurement will not be taken until the end of the time specified. This allows nodes with large caps to not appear as shorts.

ExpRes
= The expected resistance (in Ohms). This value is needed to setup the current measurement range which is based on the source voltage and the expected resistance.

EXAMPLE:
Dim RetRes As Double
Call OpenPinCheck(RetRes, 194, 0.2, 0.3, 0.001, 4700) ........................... 'Test if channel 194 is making contact with the UUT. The test will use a source voltage of 0.2 volts. The source current will be limited to less than 1.0 Amps. The test's duration will be 0.001 seconds and the expected measured resistance is 4700 Ohms.
The RCMeas functional call uses a method that quickly charges the DUT until the capacitor is charged to the source potential. Then the current through the DUT is measured and the resistance is calculated as:

\[ R = \frac{V}{I} \]

Where:
- \( V \) = source voltage
- \( I \) = measured current

A voltage ramp is then applied to the DUT and the current is measured. The capacitance is calculated as follows:

\[ C = \left( I_{TOT} - I_R \right) \frac{dv}{dt} \]

Where:
- \( I_{TOT} \) = measured current
- \( I_R \) = ramp voltage at measure time divided by the measured resistance done earlier
- \( \frac{dv}{dt} \) = source ramp

**Note:** If the “ExpCap” parameter is equal to or greater than 47uF, a two ohm resistor is placed in series with the source channel to limit the inrush current.

**Visual Basic Declaration:**

```visualbasic
Public Sub RCMeas(ByRef Capacitance As Double, ByRef Resistance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal ForcingV As Double, ByVal Limit As Double, ByVal ExpCap As Double, ByVal ExpRes As Double)
Call RCMeas(Capacitance, Resistance, Source, Measure, ForcingV, Limit, ExpCap, ExpRes)
```

**Call RCMeas(Capacitance, Resistance, Source, Measure, ForcingV, Limit, ExpCap, ExpRes)**

WHERE:

- **Capacitance** = The measured capacitance returned in Farads.
- **Resistance** = The measured resistance returned in Ohms.
Source = 0 to last available Matrix Relay channel - The channel used as a voltage source channel.

Measure = 0 to last available Matrix Relay channel - The channel used as a measurement channel.

ForcingV = -2273V to 2273V - The peak voltage must not exceed +2.5V. The peak voltage is calculated as the ramp voltage multiplied by the test time plus approximately a 10% overhead. Because the test time for this test is always .001, the peak voltage can be calculated as: \[ V_p = \text{ForcingV} \times 0.001 \times 1.1 \]

Limit = 0 to 1.0 - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 Amps, and anything else limits the source to 1.0 Amps.

ExpCap = Expected capacitance between the source and measure channels.

ExpRes = Expected resistance between the source and measure channels.

**EXAMPLE:**
```vba
Dim Capacitance As Double
Dim Resistance As Double

Call RCMeas(Capacitance, Resistance, 171, 172, 0.4, 0.3, 0.000000047, 3300)
```
```
' Measure the capacitance and resistance of the RC network.
```

Series 2040 Test System
The RCMeas2 functional call uses a method that quickly charges the DUT until the capacitor is charged to the “PeakV” potential. Then the current through the DUT is measured and the resistance is calculated as:

\[ R = \frac{V}{I} \quad \text{Where:} \quad V = \text{source voltage} \]
\[ I = \text{measured current} \]

A sinusoidal voltage is then applied to one side of the RC network in the DUT and the other side is “Guarded”. The current through the DUT is measured and the capacitance is calculated as follows:

\[ C = \frac{R \cdot I_{\text{RMS}} - V_{\text{RMS}}}{R \cdot V_{\text{RMS}} \cdot 2 \cdot \pi \cdot f} \quad \text{Where:} \quad R = \text{measured resistance from the first stage} \]
\[ I_{\text{RMS}} = \text{measured current} \]
\[ V_{\text{RMS}} = \text{source voltage} \]
\[ f = \text{source frequency} \]

Visual Basic Declaration:

Public Sub RCMeas2(ByRef Capacitance As Double, ByRef Resistance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal PeakV As Double, ByVal Limit As Double, ByVal Frequency As Double, ByVal ExpCap As Double, ByVal ExpRes As Double)
Call RCMeas2(Capacitance, Resistance, Source, Measure, PeakV, Limit, Frequency, ExpCap, ExpRes)

WHERE:

**Capacitance**

- The measured capacitance returned in Farads.

**Resistance**

- The measured resistance returned in ohms.

**Source**

- 0 to last available matrix Relay channel - The channel used as a voltage source channel.
**Measure**

\[ \text{Measure} = 0 \text{ to last available Matrix Relay channel} \]
- The channel used as a measurement channel.

**PeakV**

\[ \text{PeakV} = -2.5V \text{ to } 2.5V \]
- The peak amplitude of the applied sinusoidal source.

**Limit**

\[ \text{Limit} = 0 \text{ to } 1.0 \]
- Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 Amps, and anything else limits the source to 1.0 Amps.

**Frequency**

\[ \text{Frequency} = 0 \text{ to } 150,000 \]
- The frequency of the applied sinusoidal source.

**ExpCap**

\[ \text{ExpCap} = \text{Expected capacitance value from calculations.} \]

**ExpRes**

\[ \text{ExpRes} = \text{Expected resistance from calculations.} \]

**EXAMPLE:**

```vba
Dim Capacitance As Double
Dim Resistance As Double

Call RCMeas2(Capacitance, Resistance, 171, 172, 0.4, 0.3, 123000, 0.000000047, 3300)
```
- Measure capacitance and resistance of the RC network.
**RCMeas3**

The RCM eas3 functional call uses a method that quickly charges the DUT until the capacitor is charged to the "PeakV" potential. Then the current through the DUT is measured and the resistance is calculated as:

\[
R = \frac{V}{I} \quad \text{Where:} \quad V = \text{source voltage} \quad I = \text{measured current}
\]

A sinusoidal voltage is then applied to one side of the RC DUT. This voltage is also applied to one of the crossover detectors on the ICAM board. The current waveform through the DUT is applied to the other crossover detector on the ICAM board, and the difference in phase angle is measured. The capacitance of the DUT is calculated as follows:

\[
X_c = \frac{R}{\tan \phi} \quad \text{and} \quad C = \frac{1}{2\pi f X_c}
\]

**Difference in Phases = \phi**

**Visual Basic Declaration:**

```vbnet
Public Sub RCMeas3(ByRef Capacitance As Double, ByRef Resistance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal PeakV As Double, ByVal Limit As Double, ByVal Frequency As Double, ByVal ExpCap As Double, ByVal ExpRes As Double)
Call RCMeas3(Capacitance, Resistance, Source, Measure, PeakV, Limit, Frequency, ExpCap, ExpRes)
```

**Call RCMeas3(Capacitance, Resistance, Source, Measure, PeakV, Limit, Frequency, ExpCap, ExpRes)**

**WHERE:**

- **Capacitance** = The measured capacitance returned in Farads.
- **Resistance** = The measured resistance returned in ohms.
**Source**

\[ \text{Source} = 0 \text{ to last available Matrix Relay channel} - \text{The channel used as a voltage source channel.} \]

**Measure**

\[ \text{Measure} = 0 \text{ to last available Matrix Relay channel} - \text{The channel used as a measurement channel.} \]

**PeakV**

\[ \text{PeakV} = -2.5V \text{ to } 2.5V - \text{The peak amplitude of the applied sinusoidal source.} \]

**Limit**

\[ \text{Limit} = 0 \text{ to } 1.0 - \text{Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 Amps, and anything else limits the source to 1.0 Amps.} \]

**Frequency**

\[ \text{Frequency} = 0 \text{ to } 10,000 - \text{The frequency of the applied sinusoidal source.} \]

**ExpCap**

\[ \text{ExpCap} = \text{Expected capacitance value from calculations.} \]

**ExpRes**

\[ \text{ExpRes} = \text{Expected resistance from calculations.} \]

**EXAMPLE:**

```vbnet
Dim Capacitance As Double
Dim Resistance As Double

Call RCMeas3(Capacitance, Resistance, 171, 172, 0.4, 0.3, 10000, 0.000000047, 3300)

' Measure capacitance and resistance of the RC network.
```
**RMeas**

The RMeas functional call measures the resistance between the source channel and the measure channel. A constant voltage source is applied to the source channel for the duration of the test time, then the current is measured at the measure channel. The returned resistance is calculated by dividing the applied voltage by the measured current. An expected resistance is required to allow the measurement circuitry to adjust to the appropriate range.

**Visual Basic Declaration:**

```vbnet
Public Sub RMeas(ByRef Resistance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal SrcVoltage As Double, ByVal Limit As Double, ByVal TestTime As Double, ByVal Expected As Double)
Call RMeas(Resistance, Source, Measure, SrcVoltage, Limit, TestTime, Expected)
```

**WHERE:**

- **Resistance**
  
  The measured resistance returned in Ohms.

- **Source**
  
  0 to the last available Matrix Relay channel - The channel used as a voltage source channel.

- **Measure**
  
  0 to last available Matrix Relay channel - The channel used as a measurement channel.

- **SrcVoltage**
  
  -2.5 to +2.5 - The voltage applied to the source channel, in volts.

- **Limit**
  
  0 to 1.0 Amps - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 amp, and anything else limits the circuit to 1.0 amp.

- **TestTime**
  
  0 to 650 - The duration of the test in seconds.

- **Expected**
  
  The expected resistance, in Ohms.
EXAMPLE:
Dim RRet as Double

Call RMeas(RRet, 43, 193, 0.100, 1.0, 0.001, 10000) ............................... ‘Measure resistance ........................................................................................................ between channels 43 and 193. ............................................................................................................. Use a forcing function of 100 mV, up to 1 A for 1 mS. ........................................................................................................ Use a forcing function of 100 mV, up to 1 A for 1 mS. ........................................................................................................ Expect a resistance of about 10 KOhms ........................................................................................................ (Samples may be taken & averaged.)
The RMeas2 functional call is used to measure the resistance of small impedance components such as current sense resistors. This function sources a current through the DUT and measures it for reference. Then the function measures the voltage across the DUT through a third Matrix Relay channel which it uses to determine the effective impedance, in ohms. The DUT voltage must be supplied by one of the system Isolation Amplifiers, amplifying or buffering the differential voltage across the DUT. Any gain added by the Isolation Amplifier must be taken into account on the returned reading.

Visual Basic Declaration:
```
Public Sub RMeas2(ByRef Resistance As Double, ByVal SourceCh As Integer, ByVal MeasureICh As Integer, ByVal MeasureVCh As Integer, ByVal ForcingI As Double, ByVal Limit As Double, ByVal TestTime As Double, ByVal ExpRes As Double)
Call RMeas2(Resistance, SourceCh, MeasureICh, MeasureVCh, ForcingI, Limit, TestTime, ExpRes)
```

Call RMeas2(Resistance, SourceCh, MeasureICh, MeasureVCh, ForcingI, Limit, TestTime, ExpRes)
WHERE:

**Resistance**
\[ = \text{The measured resistance returned in Ohms.} \]

**SourceCh**
\[ = 0 \text{ to the last available Matrix Relay channel - The channel used as a current source channel.} \]

**MeasureICh**
\[ = 0 \text{ to the last available Matrix Relay channel - The channel used to measure the current through the DUT.} \]

**MeasureVCh**
\[ = 0 \text{ to the last available Matrix Relay channel - The channel used to measure the output voltage of the Isolation Amplifier.} \]

**Forcingl**
\[ = \text{Determines the DC current source magnitude, voltage-limited by the Limit parameter. Because of the small voltages involved in the measurements, it is advantageous to source the largest current possible on the present system (50mA). This will help to reduce the impact of noise in the system and require less gain from the Isolation Amplifiers to provide the best possible readings.} \]

**Limit**
\[ = 3.0V \text{ to 35.0V - The voltage limit on the current source (in Volts).} \]

**TestTime**
\[ = 0 \text{ to 650 - The duration of the test in seconds. Note that some settling time may be required if the Isolation Amplifier 160Hz filter is used.} \]

**ExpRes**
\[ = \text{The expected resistance, in Ohms.} \]
EXAMPLE:

Dim RRet as Double

Call AuxRly(0, 1) ................................................................. 'Close Aux Relays.
Call AuxRly(1, 1)

Call Inst(0, 2, 1) ............................................. 'Program the IsoAmp for a gain of 100, 16KHz filter.
......................................................................................... (Slower filters may require longer test times.)

Call RMeas2(RRet, 222, 229, 225, 0.50, 5.0, 0.001, 0.010)
............................................................................. 'Call the measurement function for a 50mA source @ 1mS duration.

RRet = RRet/100 ......................................................... 'Adjust the returned value by the gain.

Call AuxRly(0, 0) ............................................................................. 'Disconnect the IsoAmp.
Call AuxRly(1, 0)
The RofRCMeas functional call uses a method that quickly charges the DUT until the capacitor is charged to the “ForcingV” voltage divided by 1000. This method maintains consistency with the CofRCMeas and RCMeas functional calls. This call assumes that the value of the parallel capacitance being passed in the “ExpCap” parameter is reasonable. The current through the DUT is then measured, and the resistance is calculated as:

\[ R = \frac{V}{I} \]

Where: 
- \( V \) = source voltage  
- \( I \) = measured current

Visual Basic Declaration:
```visualbasic
Public Sub RofRCMeas(ByRef Resistance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal ForcingV As Double, ByVal Limit As Double, ByVal ExpCap As Double, ByVal ExpRes As Double)
Call RofRCMeas(Resistance, Source, Measure, ForcingV, Limit, ExpCap, ExpRes)
```

**WHERE:**

- **Resistance**  
  The measured resistance returned in ohms.

- **Source**  
  0 to last available Matrix Relay channel - The channel used as a voltage source channel.

- **Measure**  
  0 to last available Matrix Relay channel - The channel used as a measurement channel.

- **ForcingV**  
  -2273V to 2273V - The DUT will be charged to this potential divided by 1000. This parameter is handled in this way to stay consistent with CofRCMeas and RCMeas.

- **Limit**  
  0 to 1.0 - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 Amps, and anything else limits the source to 1.0 Amps.
**ExpCap**

= Expected capacitance between the source and measure channels.

**ExpRes**

= Expected resistance between the source and measure channels.

**EXAMPLE:**

Dim RetRes As Double

Call RofRCMeas(RetRes, 171, 172, 0.4, 0.3, 0.000000047, 3300)

 measure the resistance of the RC network.
The RofRCMeas2 functional call uses a method that quickly charges the DUT up to the “PeakV” voltage and then the current is measured. This call assumes that the value of parallel capacitance passed in the “ExpCap” parameter is reasonable. The resistance is then calculated as:

\[ R = \frac{V}{I} \]

Where:
- \( V \) = source voltage
- \( I \) = measured current

**Visual Basic Declaration:**

```vbnet
Public Sub RofRCMeas2(ByVal Resistance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal PeakV As Double, ByVal Limit As Double, ByVal Frequency As Double, ByVal ExpCap As Double, ByVal ExpRes As Double)
Call RofRCMeas2(Resistance, Source, Measure, PeakV, Limit, Frequency, ExpCap, ExpRes)
```

**WHERE:**

- **Resistance**
  - The measured resistance returned in ohms.

- **Source**
  - 0 to last available Matrix Relay channel - The channel used as a voltage source channel.

- **Measure**
  - 0 to last available Matrix Relay channel - The channel used as a measurement channel.

- **PeakV**
  - -2.5V to 2.5V - DC voltage the DUT is charged to before measuring the resistor current.

- **Limit**
  - 0 to 1.0 - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 Amps, and anything else limits the circuit to 1.0 Amps.
**Functional Calls**

**Frequency**
= This parameter is unused for this function. It is kept to be consistent with the RCM eas2 and CofRCMeas2 functional calls.

**ExpCap**
= Expected capacitance value from calculations.

**ExpRes**
= Expected resistance from calculations.

**EXAMPLE:**

```plaintext
Dim Resistance As Double

Call RofRCM eas(Resistance, 171, 172, 0.4, 0.3, 123000, 0.000000047, 3300)
............................................................................ 'Measure the resistance of the RC network.
```
The RofRCMeas3 functional call uses a method that quickly charges the DUT up to the “PeakV” voltage and then the current is measured. This call assumes that the value of parallel capacitance passed in the “ExpCap” parameter is reasonable. The resistance is then calculated as:

\[ R = \frac{V}{I} \]

Where:
- \( V \) = source voltage
- \( I \) = measured current

Visual Basic Declaration:
```
Public Sub RofRCMeas3(ByRef Resistance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal PeakV As Double, ByVal Limit As Double, ByVal Frequency As Double, ByVal ExpCap As Double, ByVal ExpRes As Double)
```

WHERE:

- **Resistance** = The measured resistance returned in ohms.
- **Source** = 0 to last available Matrix Relay channel - The channel used as a voltage source channel.
- **Measure** = 0 to last available Matrix Relay channel - The channel used as a measurement channel.
- **PeakV** = -2.5V to 2.5V - DC voltage the DUT is charged to before measuring the resistance current.
- **Limit** = 0 to 1.0 - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 Amps, and anything else limits the circuit to 1.0 Amps.
**Functional Calls**

**Frequency**

= This parameter is unused for this function. It is kept to be consistent with the RCM eas3 and CofRCM eas3 functional calls.

**ExpCap**

= Expected capacitance value from calculations.

**ExpRes**

= Expected resistance from calculations.

**EXAMPLE:**

```vbnet
Dim Resistance As Double
Call RofRCMeas3(Resistance, 171, 172, 0.4, 0.3, 10000, 0.000000047, 3300)
```

………………………………………………………………………………………………….. Measure the resistance of the RC network.
ShortsMeas

The ShortsMeas functional call executes a shorts test on a channel. Typically all of the channels will be guarded with GuardBd prior to executing any shorts tests. ShortsMeas will remove the guard on the channel under test. A constant voltage is applied to the node for the given test time and the source’s output current is measured. The guard is restored to the source channel and the calculated resistance is returned where the user can compare it to an acceptable value.

Visual Basic Declaration:
Public Sub ShortsMeas(ByRef Resistance as Double, ByVal Channel As Integer, ByVal SrcVoltage As Double, ByVal Limit As Double, ByVal TestTime As Double)
Call ShortsMeas(Resistance, Channel, SrcVoltage, Limit, TestTime)

WHERE:

Resistance = Return variable - The measured resistance of the node to the guard bus.

Channel = 0 to last available Matrix Relay channel - The channel used to test for a short.

SrcVoltage = -2.5 to +2.5 - The voltage applied to the channel. Parameter is in volts.

Limit = 0.0 to 1.0 - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 amps. Anything else sets the limit to 1.0 amp.

TestTime = 0 to 650 - The duration of the test in seconds. The measurement will not be taken until the end of the time specified. This allows nodes with large caps to not appear as shorts.
**EXAMPLE:**

Dim RRet as Double

Call ShortsMeas(RRet, 55, 0.100, 0.200, 0.001) ............... ‘Test channel 55 for node resistance.  
.............................................................. Use a forcing function of 100 mv, up to 0.2 amps for 1 mS.
The SKelvin functional call is used to enable or disable the source Kelvin capability of the ICAM board and selects the Matrix Relay channel to be used. If the channel number specified is positive, the ICAM board source Kelvin is enabled and routed to that Matrix Relay channel. If the channel parameter is negative, the ICAM board’s source Kelvin capability is disabled and the Matrix Relay channel used for source Kelvin is reset (if previously used).

Visual Basic Declaration

Public Sub SKelvin(ByVal Channel As Integer)
Call SKelvin(Channel)
WHERE:

Channel

= < 0 Disconnects previously set SKelvin channel and resets the source Kelvin circuit.
= 0 to the last available channel = the channel to connect to SKelvin bus (M RLY bus 2).

EXAMPLE:

Call SKelvin(-1) ....................................................... 'Disable the ICAM source Kelvin capability and disconnect the Matrix Relay channel previously connected as the source Kelvin channel.

Call SKelvin(520) .......................................... 'Connect MRly channel 520 to source Kelvin bus and enable the ICAM Source Kelvin capability.
SMux

The SMux functional call is used to provide an auxiliary source signal to any Matrix Relay channel through one of the ICAM buses. Each of the four buses in the ICAM board may be disengaged from its default MDA output (or input) signal, and switched to one of its auxiliary source inputs. Then a connection will be made through the bus to the specified Matrix Relay channel to provide the source signal to the fixture. The MDA bus signals are:

0 Measurement Bus (TPB34)
1 Voltage Source Bus (BPB34)
2 Current Source Bus (BPB33)
3 Guard Bus (TPB33)

Note: The auxiliary source is disconnected if SMux is followed by a later MMux call to the same bus or a component measurement is taken.

Visual Basic Declaration:

Public Sub SMux(ByVal Input As Integer, ByVal Bus As Integer, ByVal Channel As Integer)

WHERE:

Input
= -1 Reset a single bus (specified below).
= 0 to 3 indicates the Bus input signal to select.

Bus
= -1 Reset all buses (to default MDA signal).
= 0 to 3 indicates the Bus used to connect input signal to channel output.

Channel
= 0 to the last available Matrix Relay channel.

EXAMPLE:

Call SMux(0, -1, 0).................. Reset all SMux inputs, buses & channels.
Call SMux(-1, 2, 0).............. Reset bus 2, its input, and the channel it was connected to.
Call SMux(2, 1, 98).................. Route SMux input 2 on bus 1 to channel 98.
TPol

The TPol functional call sources a voltage ramp into one side of the primary coil of a transformer and guards the other side. Then the finishing voltage of the induced voltage ramp is measured by connecting the ICAM RMUX circuitry to the secondary terminal that is in phase with the side of the primary being driven and guarding the other side of the secondary.

When the surrounding circuitry complies, it is possible to calculate the turns ratio by dividing the finishing source voltage by the measured voltage.

Visual Basic Declaration:

```vbnet
Public Sub TPol(ByRef Voltage As Double, ByVal PPlus As Integer, ByVal PMinus As Integer, ByVal SPlus As Integer, ByVal SMinus As Integer, ByVal RampV As Double, ByVal Limit As Double, ByVal TestTime As Double, ByVal ExpVolts As Double)
Call TPol(Voltage, PPlus, PMinus, SPlus, SMinus, RampV, Limit, TestTime, ExpVolts)
```

WHERE:

- **Voltage** = The measured voltage returned in Volts.
- **PPlus** = 0 to the last available Matrix Relay channel - The channel of the primary coil node selected to drive the voltage source.
- **PMinus** = 0 to the last available Matrix Relay channel - The channel used for the remaining primary coil lead.
- **SPlus** = 0 to the last available Matrix Relay channel - The channel of the secondary coil node that corresponds in phase to the sourced channel of the primary coil.
- **SMinus** = 0 to the last available Matrix Relay channel - The channel used for the remaining secondary coil lead.
**Functional Calls**

**RampV**

= 0 to 1800 Volts/Second at a test time of 1 millisecond. Ramp voltage used to excite the transformer primary.

**Limit**

= 0 to 1.0 Amps - Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 amp, and anything else limits the circuit to 1.0 amp.

**TestTime**

= 0 to 650 - The duration of the test in seconds.

**ExpVolts**

= The expected voltage.

**EXAMPLE:**

```vbnet
Dim Voltage As Double

Call ICAMSamples(1)
Call ICAMDelay(9) ................ ' Set up measurement to be one sample at 90% of the test time.

Call TPOL(Voltage, 101, 152, 106, 108, 400, 0.2, 0.001, 0.2) ..... ' Measure the finishing voltage of the ramp induced on the secondary. Compare returned voltage to predetermined limits and report pass/fail
```
The TwoSrcDVMeas functional call measures the voltage drop across a forward-biased p-n junction. An external voltage may be routed through an SMux bus 1 input for user-defined purposes such as gating an SCR. A constant current is applied to the source channel (anode) for the duration of the test time, and the cathode will be connected to the Guard bus. The returned voltage is measured at the source, and will equal the forward voltage drop of the p-n junction.

This call is functionally and parametrically identical to the DVMeas functional call with one exception: The voltage source bus is not cleared. This is an advantage when a specific test requires two sources, such as the SCR test. In this example, one source would be required to gate the SCR and the other to provide the current sourced when measuring the voltage generated across the SCR. The SMux call must be used prior to the TwoSrcDVMeas call to route the second voltage source onto the ICAM board. SMux bus 1 is the only bus that can be used for this purpose.

Visual Basic Declaration:

```vbnet
Public Sub TwoSrcDVMeas(ByRef Voltage As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal SrcCurrent As Double, ByVal Limit As Double, ByVal TestTime As Double, ByVal Expected As Double)
Call TwoSrcDVMeas(Voltage, Source, Measure, SrcCurrent, Limit, TestTime, Expected)
```

WHERE:

- **Voltage**
  - The measured voltage returned in Volts.
- **Source**
  - 0 to the last available Matrix Relay channel - The channel used as a current source channel.
- **Measure**
  - 0 to last available Matrix Relay channel - The channel used as a guard.
**SrcCurrent**

\[ \text{SrcCurrent} = 0 \text{ to } 50\text{mA} \] - The current applied to the source channel, in amps.

\[ \text{Limit} = 2.0\text{V} \text{ to } 35\text{V} \] - The voltage limit on the current source in volts.

\[ \text{TestTime} = 0 \text{ to } 650 \] - The duration of the test in seconds.

\[ \text{Expected} = \] The expected return voltage in volts.

**EXAMPLE:**

```vbnet
dim VRet as Double

Call TwoSrcDVMeas(VRet, 0, 1, 0.02, 2, 0.001, 0.7) 'Measure the forward-biased voltage drop of a p-n junction from channel 0 to channel 1. Use a current source of 20mA, and set the compliance voltage to 2 volts. Use a test time of 1mS.
```
TwoSrcRMeas

The TwoSrcRMeas functional call measures the resistance between the source channel and the measure channel. A constant voltage source is applied to the source channel for the duration of the test time, then the current is measured at the measure channel. The returned resistance is calculated by dividing the applied voltage by the measured current. An expected resistance is required to allow the measurement circuitry to adjust to the appropriate range.

This call is functionally and parametrically identical to the RMeas functional call with one exception: SMux bus 2 (the current source bus) is not cleared. This is an advantage when a specific test requires two sources, such as the closed contact resistance of a normally open relay. In this example, one source would be required to activate the coil and the other to provide the voltage sourced when measuring the contact resistance. The SMux call must be used prior to the TwoSrcRMeas call to route the second voltage source onto the ICAM board. SMux bus 2 is the only bus that can be used for this purpose.

Visual Basic Declaration:

```vba
Public Sub TwoSrcRMeas(ByRef Resistance As Double, ByVal Source As Integer, ByVal Measure As Integer, ByVal SrcVoltage As Double, ByVal Limit As Double, ByVal TestTime As Double, ByVal Expected As Double)
```

WHERE:

- **Resistance** = The measured resistance returned in Ohms.
- **Source** = 0 to the last available Matrix Relay channel - The channel used as a voltage source channel.
- **Measure** = 0 to last available Matrix Relay channel - The channel used as a measurement channel.
- **SrcVoltage** = -2.5 to +2.5 - The voltage applied to the source channel, in volts.
**Functional Calls**

- **Limit**
  - Value: 0
  - Description: Current limit of the voltage source. Values less than or equal to 0.2 fix the limit at 0.2 amp, and anything else limits the circuit to 1.0 amp.

- **TestTime**
  - Value: 0
  - Description: The duration of the test in seconds.

**EXAMPLE:**

```vbnet
Dim RRet as Double

Call TwoSrcMeas(RRet, 43, 193, 0.100, 1.0, 0.001, 10000) ' Measure resistance between channels 43 and 193. Use a forcing function of 100 mV, up to 1 A for 1 mS. Expect a resistance of about 10 KOhms (Samples may be taken & averaged.)
```
UnGuard

The UnGuard functional call clears a single Matrix Relay channel from the Guard Bus.

**NOTE:** The UnGuard functional call is a “deferred” call. The UnGuard function will not take place until an “immediate” call such as a measurement call is invoked.

**Visual Basic Declaration:**
Public Sub UnGuard(ByVal Channel As Integer)

**Call UnGuard (Channel)**

**WHERE:**

```
Channel = 0 to the last available Matrix Relay channel.
```

**EXAMPLE:**

```
Call UnGuard(83)..................................................'Disconnect channel 83 from the guard bus.
```
The UnGuardBd functional call disconnects all of the Matrix Relay channels on a given board from the Guard Bus.

**NOTE:** The UnGuardBd functional call is a “deferred” call. The UnGuardBd function will not take place until an “immediate” call such as a measurement call is invoked.

**Visual Basic Declaration:**

```
Public Sub UnGuardBd(ByVal MrlyBoardNum As Integer)
Call UnGuardBd(MrlyBoardNum)
```

**WHERE:**

```
MrlyBoardNum
= Board number whose channels are to be disconnected from the Guard Bus.
= 0  Mrly channels 0 to 63
= 1  Mrly channels 64 to 127, etc. to 15.
```

**EXAMPLE:**

```
Call UnGuardBd(3) ............................... ’Disconnect matrix Relay channel numbers 192 to 255
............................................................................................................... from the Guard Bus.
```
UnVSrcBd

The UnVSrcBd functional call disconnects all of the Matrix Relay channels on a given board from the Voltage Source Bus.

**NOTE:** The UnVSrcBd functional call is a “deferred” call. The UnVSrcBD function will not take place until an “immediate” call such as a measurement call is invoked.

**Visual Basic Declaration:**

```vbnet
Public Sub UnVSrcBd(ByVal MrlyBdNum As Integer)
    Call UnVSrcBd(MrlyBdNum)
End Sub
```

**WHERE:**

- **MrlyBoardNum**
  - = Board number whose channels are to be disconnected from the Voltage Source Bus.
  - = 0 Mrly channels 0 to 63.
  - = 1 Mrly channels 64 to 127, etc. to 15.

**EXAMPLE:**

```vbnet
Dim MrlyBdNum As Integer

For MrlyBdNum = 3 To 4
    Call UnVSrcBd(MrlyBdNum)
Next MrlyBdNum 'Disconnect Matrix Relay channel numbers 192 to 319 from the Voltage Source Bus.
```

---

**ICAM User Manual V3.0**

**Series 2040 Test System**
VMeas

The VMeas functional call is provided to take voltage measurements through the signal inputs specified by the HFMux and MMux functional calls using a “-1” as their Sig parameter. The value returned is an average of the number of samples specified using the ICAM Samples functional calls. The sampling frequency is determined by the specified TestTime, the ICAM Delay, and the number of samples. If the samples used to calculate the average are required for analysis, they may be retrieved using the ICAM GetSamples functional call.

An explanation of the timing of an ICAM measurement is included at the end of this section.

Visual Basic Declaration:
Public Sub VMeas(ByRef Volts As Double, ByVal Source As Integer, ByVal TestTime As Double)

WHERE:

Volts = Voltage average of samples taken.

Source = 0 to 3. ICAM bus to use to get a measurement sample.

TestTime = 0 to 650 sec. Duration of the test (in Seconds).

EXAMPLE:
Dim VRet as Double
Call VMeas(VRet, 2, 0.001) .................................. ‘Start Voltage Measurement System sampling
........................................................................ on ICAM bus 2. Take preprogrammed number
................................................................................................................ of samples over 1 millisecond.
Call VMeas(VRet, 3, 1.0) ............................. ‘Start Voltage Measurement System sampling
...................................................................................... on ICAM bus 3. Take preprogrammed number
................................................................................................................ of samples over 1 second.
**VSrcBd**

The VSrcBd functional call connects all of the Matrix Relay channels on a given board to the Voltage Source Bus.

**NOTE:** The GVSrcBd functional call is a “deferred” call. The VSrcBd function will not take place until an “immediate” call such as a measurement call is invoked.

**Visual Basic Declaration:**

```
Public Sub VSrcBd(ByVal MrlyBoardNum As Integer)
Call VSrcBd(MrlyBoardNum)
```

WHERE:

```
MrlyBoardNum
   = Board number whose channels are to be connected to the Voltage Source Bus.
   = 0   Mrly channels 0 to 63.
   = 1   Mrly channels 64 to 127, etc. to 15.
```

**EXAMPLE:**

```
Call VSrcBd(0).................................................'Connect Matrix Relay channel numbers 0 to 63 to the Voltage Source Bus.
```
ICAM Timing Issues

Test Time - This parameter is given with a measurement functional call, and is the total length of time that the sourcing function is applied to the DUT, and the time in which all samples must be taken.

Test Delay - This parameter is given using the ICAM Delay functional call, and is the fraction of the Test Time that the sourcing function is applied before any samples are taken.

Samples - This parameter is given using the ICAM Samples functional call, and is the total number of samples taken by the measurement system. All samples are taken between the end of the Test Delay and before Test Time runs out.

Returning Values - All values are returned in the whole units described in the specific functional calls, and are averaged if more than one sample is taken.

Retrieving Samples - If further analysis or graphing of the samples is needed, ICAM GetSamples may be used to download the contents of the sample memory. The values are returned in whole Volts or Amps, depending on the technique used by the measurement function called.

Bad Sampling Frequencies - If a bad sample frequency error is returned from a measurement functional call, it indicates that too many samples are requested for the time allotted for them. The sampling frequency would need to be higher than the hardware is capable of (12.5 KHz, max). There are three options to solve this problem:

1) Decrease the number of samples (ICAM Samples)
2) Increase the Test Time (in the measurement call)
3) Decrease the Test Time Delay (ICAM Delay)
Note: Because the delay circuit is independent of the sampling frequency, the first sample may not be taken precisely at the end of the delay. However, the last sample is guaranteed to be taken before the end of the test time. If this effect is undesirable, minimize it by increasing the number of samples taken and thus increase the sampling frequency. Presently, the maximum sampling frequency is 12.5 KHz.

Sampling frequency = \( \frac{\text{Number of samples}}{\text{Test Time (1 - Delay)}} = \frac{5}{.01(1 - .75)} = 2\text{kHz} \)

Sampling frequency = \( \text{Test Time (1 - Delay)} = \frac{5}{.01(1 - .75)} = 2\text{kHz} \)

Delay (Parameter given in ICAMdelay) (Example = .75)

Actual Measurements taken during this segment:

NSamples (Parameter in ICAMSamples) (Example = 5)

Test Time (Parameter given in the *Meas* functional calls) (Example = 10ms)
Test Descriptions
Test Descriptions

The following pages contain brief descriptions on the structure of the tests for each of the measurement functional calls. These tests are as follows:

- **AltSrcRMeas** - Alternate source resistor measurement
- **BetaMeas** - Transistor Beta Measurement
- **CDischarge** - Capacitor discharge test
- **CMeas** - Capacitance measurement
- **CofRCMeas** - Capacitance measurement of a RC network using a voltage ramp for capacitance
- **CofRCMeas2** - Capacitance measurement of a RC network using a sinusoidal waveform for capacitance
- **CofRCMeas3** - Capacitance measurement of a RC network using a sinusoidal waveform to measure the phase shift for capacitance
- **CPol** - Capacitor polarity test
- **DVMeas** - Voltage drop across a forward biased p-n junction
- **DZMeas** - Voltage drop across a reverse biased Zener diode
- **LMeas** - Inductance measurement
- **OpenPinCheck** - Executes an opens test on a given channel.
- **RCMeas** - RC network tests using a voltage ramp for capacitance
- **RCMeas2** - RC network tests using a sinusoidal waveform for capacitance
- **RCMeas3** - RC network tests using a sinusoidal waveform for capacitance
- **RMeas** - Resistance measurement
- **RofRCMeas** - Resistance measurement of an RC network with a time delay to allow the capacitor to charge
- **RofRCMeas2** - Resistance measurement of an RC network with a time delay to allow the capacitor to charge
**RofRCMeas3** - Resistance measurement of an RC network with a time delay to allow the capacitor to charge

**ShortsMeas** - Executes a shorts test on a given channel

**TwoSrcDVMeas** - Measures voltage across a forward-biased p-n junction

**TwoSrcRMeas** - Used for resistance tests requiring two sources

Signal paths for the multiplexer functional calls are also shown to include:

**HFMux** - Routes a high-frequency input to the ICAM Voltage Measurement System or the AMS board

**MMux** - Routes an ICAM channel to the ICAM Voltage Measurement System or the AMS board

**SMux** - Routes an auxiliary source signal to any ICAM channel through one of the ICAM buses
The `AltSrcRMeas` functional call is used to measure resistance by using an external source. When using this function, it is important to restrict the external source to 15 volts because if the DUT was shorted (a short circuit), the source voltage could compromise the ICAM current measurement amplifier.
The BetaMeas functional call executes a beta measurement on a transistor. A voltage is applied from an external source to the transistor’s collector, a constant current is applied to the base, and the emitter current is measured. Since a transistor’s beta is defined as the collector current ($I_c$) divided by the base current ($I_b$) and the collector current is calculated as the emitter current ($I_e$) minus the base current, the transistor’s beta is calculated by $\frac{I_e - I_b}{I_b}$.
The CDischarge functional call is used to dissipate a charge present across a capacitor to less than a specified voltage level. The function relies on an external discharge resistor (in the fixture) to shunt and limit the current generated by the discharge. The resistor is connected between Analog ground and an ICAM SMux input on buses 0, 1, or 2.

The functional call applies the external resistor across the capacitor and begins monitoring the capacitor voltage. When it is less than the predetermined value, the load is removed and the function exits. If the timeout elapses before the value is reached, the function returns immediately with an error indicating this condition.

Caution: Guards should not be used with this functional call.
CAPACITOR DISCHARGE PROCEDURE

TOTAL TEST TIME: Approximately 50 milliseconds overhead + discharge time (calculated from the size of capacitor C, size of the discharge resistor R, and the acceptable limit voltage).

METHOD: The functional call CDischarge is used to dissipate a charge present across a capacitor to less than a specified voltage level. The function relies on an external discharge resistor (in the fixture) to shunt and limit the current generated by the discharge.

The functional call applies the external resistor across the capacitor and begins monitoring the capacitor voltage. When it is less than the selected threshold, the load is removed and the function exits. If the timeout elapses before the value is reached, the function returns prematurely with an error indicating this condition.

The only additional hardware required consists of a single discharge load, R, selected by two criteria. First, the load should be as small as possible to allow for a rapid discharge to minimize test time. Second, the load must be large enough to limit the current passing through the ICAM and Matrix Relay boards. The switches on these boards are capable of reliably switching 500mA before the relay contacts begin to degrade.

FIXTURE: The two nodes on either side of the capacitor need to be connected to two Matrix Relay channels dedicated to the ICAM board’s measurement signals. These connections may already be in place to allow component measurement of the capacitor using the CMeas function. If this is the case, then these connections are suitable for discharge purposes as well.

The external load resistor should be connected between an analog ground pin (AGND) and an SMux input (buses 0, 1 or 2 only.) This input pin is specified in the function call’s parameter list. If additional loads of varying sizes are required for multiple capacitors, they may be connected to additional SMux inputs, or multiplexed to a single input, as desired. Additional capacitors each require an individual CDischarge call.

The following example code uses a load resistor connected between analog ground (right pin of row 12 of the ICAM slot) and Smux bus 2, input 0 (left pin of row 18 of the ICAM slot).
**USAGE:** The functional call declaration:

```vbnet
Public Sub CDischarge(ByVal HiChan As Integer, ByVal LoChan As Integer, ByVal Bus As Integer, ByVal SMuxInput As Integer, ByVal limit As Double, ByVal timeout As Double)
```

The HiChan and LoChan parameters are the channel numbers of the two networks connected to the capacitor to be discharged. (They are named “HiChan” and “LoChan” for reference purposes only; if reversed, the function will discharge the cap until the magnitude of the charge is below the specified limit.)

The Bus and SMuxInput parameters specify the ICAM SMux input that the load is attached to.

The function ends when the magnitude of the capacitor charge is less than the limit, or when the timeout is reached. If the timeout occurs, the function will return an error (101:61), indicating that the capacitor is not fully discharged. The timeout must be chosen with the maximum discharge time in mind. When choosing a timeout, using a result value of $5R\times C$ will discharge the capacitor 99%. If possible, double the timeout value because it doesn’t add any test time to a passing test, and it allows for component tolerance and fixture variances.

**EXAMPLE CODE:**

```
'***** Discharge a 35 Volt 5000uF cap across channels 66 and 67, through a 100 ohm resistor connected to SMux bus 2, input 0 *****

'***** 100 mV limit, 3.5 second timeout (2.5 seconds calculated discharge time) *****

Call CDischarge (66, 67, 2, 0, 0.1, 3.5)
```

**CONCERNS:**

All switching circuitry is limited to 500mA of current, 200V.

Large capacitors, especially electrolytic capacitors, recover somewhat after discharge to regain a portion of their former voltage.
CMeas

The CMeas functional call measures the capacitance between the source channel and the measure channel. A voltage ramp is applied to the source channel for the duration of the test time, then the current is measured at the measure channel. The returned capacitance is calculated by dividing the measured current by the applied voltage ramp. An expected capacitance is required to allow the measurement circuitry to adjust to the appropriate range.

NOTE: If the “Expected” parameter is equal to or greater than 47μF, a two ohm resistor is placed in series with the source channel to limit the inrush current.

Call CMeas(Capacitance, Source, Measure, SrcVoltage, Limit, TestTime, Expected)
The CPol functional call sources a voltage into the positive lead of a capacitor and measures the leakage current present on the negative lead of the capacitor. If the capacitor is incorrectly installed, the leakage current through the capacitor will be significantly higher than the leakage current found to be nominal during test development.

The CDIscharge function should be run after the test to remove any charge which may still be present on the capacitor.

NOTE: This test requires the use of an external voltage source connected to a SMux Bus Input (Buses 1, 2, or 3).

Call CPol(Current, PosChan, NegChan, SMuxBus, SMuxInput, TestTime, ExpCurr)
CofRCMeas

The CofRCMeas functional call applies a voltage ramp to the DUT and the resulting current is measured. It assumes that the value being passed in the "ExpRes" parameter is valid. The capacitance is calculated as:

\[ C = \left( I_{TOT} - I_{RES} \right) \frac{dv}{dt} \]

Where:
- \( I_{TOT} \) = measured current
- \( I_{RES} \) = calculated current due to resistor
- \( \frac{dv}{dt} \) = source ramp

Call CofRCMeas(Capacitance, Source, Measure, ForcingV, Limit, ExpCap, ExpRes)
The CofRCMeas2 functional call applies a sinusoidal voltage to one side of the RC and the other side is “Guarded”. This function assumes that the value of parallel resistance being passed in the “ExpRes” parameter is valid. The current through the DUT is measured and the capacitance is calculated as follows:

\[
C = \frac{R \cdot I_{RMS} - V_{RMS}}{R \cdot V_{RMS} \cdot 2 \cdot \pi \cdot f}
\]

Where:
- \( R = \) ExpRes
- \( I_{RMS} = \) Measured current
- \( V_{RMS} = \) Source voltage
- \( f = \) Source frequency

**Caution:**
Guards should not be used with this functional call.
The CofRCMeas3 functional call applies a sinusoidal voltage to one side of the RC DUT. This voltage is also applied to one of the crossover detectors on the ICAM board. The current waveform through the DUT is applied to the other crossover detector on the ICAM board, and the difference in phase angle is measured. The capacitance of the DUT is calculated as follows:

\[ X_c = \frac{R}{\tan\phi} \quad \& \quad C = \frac{1}{2\pi f X_c} \]

**Call CofRCMeas3(Capacitance, Source, Measure, PeakV, Limit, Frequency, ExpCap, ExpRes)**

Caution: Guards should not be used with this functional call.
The `DVMeas` function measures the voltage drop across a forward biased p-n junction. A constant current is applied to the source channel (anode) for the duration of the test time, and the cathode will be connected to a ground channel. The returned voltage is measured at the source, and will equal the forward voltage drop of the p-n junction.

Call `DVMeas(Voltage, Source, Measure, SrcCurrent, Limit, TestTime)`
The DZMeas function measures the voltage drop across a reverse biased Zener diode. A constant current is applied to the source channel (cathode) for the duration of the test time, and the anode should be connected to the measurement channel. The returned voltage is measured at the source, and should equal the reverse breakdown voltage of the p-n junction.

Call DZMeas(Voltage, Source, Measure, SrcCurrent, Limit, TestTime)
The LMeas functional call measures the inductance between the source channel and the measure channel. A sinusoidal voltage is applied to the source channel with the measurement channel grounded. The source current is measured to determine the impedance of the component at the given frequency. The inductance may then be calculated from the impedance and source frequency.

\[ L = \frac{V}{2\pi f I} \]

**Call LMeas(Inductance, Source, Measure, SrcVoltage, Limit, Frequency, Expected)**

Caution: Guards should not be used with this functional call.
The OpenPinCheck functional call executes an opens test on the given channel. This test is used to determine if a fixture pogo pin is in fact making contact with the UUT. Typically all of the channels will be connected to the voltage source bus by using the VSrcBd functional call prior to executing any opens test. OpenPinCheck will disconnect the given channel from the voltage source bus and connect it to the measurement bus. The given constant voltage will be applied to all of the nodes connected to the voltage source bus. Then the current is measured at the measurement node. This current should be equal to the source voltage divided by the parallel resistance connected to the measurement node. The calculated parallel resistance is returned to the user. For further information, see the section on ShortsMeas on page xxx.
RCMeas

1. MRLY Source → DUT → MRLY Measure
   - $I_e = 0$ when charged

2. MRLY Source → DUT → MRLY Measure

3. MRLY Source → DUT → MRLY Measure
   - $I_{TO}$

**Call RCMeas(Capacitance, Resistance, source, measure, ForcingV, Limit, ExpCap, ExpRes)**

The RCMeas functional call uses a method that quickly charges the DUT until the capacitor is charged to the source potential. Then the current through the DUT is measured and the resistance is calculated as:

$$R = \frac{V}{I}$$

Where:
- $V$ = source voltage
- $I$ = measured current

When the resistance is calculated, the capacitor is quickly discharged.

A voltage ramp is then applied to the DUT and the current is measured. The capacitance is calculated as follows:

$$C = (I_{TOT} - I_R) \frac{dv}{dt}$$

Where:
- $I_{TOT} =$ measured current
- $I_R = \text{is ramp voltage at measure time divided by the measured resistance done earlier}$
- $\frac{dv}{dt} = \text{source ramp}$

Caution: Guards should not be used with this functional call.
NOTE: If the “Expected” parameter is equal to or greater than 47μF, a two ohm resistor is placed in series with the source channel to limit the inrush current.
The RCMeas2 functional call uses a method that quickly charges the DUT until the capacitor is charged to the “PeakV” potential. Then the current through the DUT is measured and the resistance is calculated as:

\[
R = \frac{V}{I}
\]

Where:
- \( V \) = source voltage
- \( I \) = measured current

When the resistance is calculated, the capacitor is quickly discharged.

A sinusoidal voltage is then applied to one side of the RC network and the other side is “Guarded”. The current through the DUT is measured and the capacitance is calculated as follows:

\[
C = \frac{R \cdot I_{\text{RMS}} - V_{\text{RMS}}}{R \cdot V_{\text{RMS}} \cdot 2 \pi f}
\]

Where:
- \( R \) = Measured resistance from the first stage
- \( I_{\text{RMS}} \) = Measured current
- \( V_{\text{RMS}} \) = Source voltage
- \( f \) = Source frequency

Caution:
Guards should not be used with this functional call.
The RCMeas3 functional call uses a method that quickly charges the DUT until the capacitor is charged to the “PeakV” potential. Then the current through the DUT is measured and the resistance is calculated as:

\[ R = \frac{V}{I} \]

Where:
- \( V \) = source voltage
- \( I \) = measured current

When the resistance is calculated, the capacitor is quickly discharged.

A sinusoidal voltage is then applied to one side of the RC DUT. This voltage is also applied to one of the crossover detectors on the ICAM board. The current waveform through the DUT is applied to the other crossover detector on the ICAM board, and the difference in phase angle is measured. The capacitance of the DUT is calculated as follows:

Call RCMeas3(Capacitance, Resistance, Source, Measure, PeakV, Limit, Frequency, ExpCap, ExpRes)
\[ X_c = \frac{R}{\tan \phi} \quad \text{and} \quad C = \frac{1}{2\pi f X_c} \]

Caution:
Guards should not be used with this functional call.
The RMeas functional call measures the resistance between the source channel and the measure channel. A constant voltage source is applied to the source channel for the duration of the test time, then the current is measured at the measure channel. The returned resistance is calculated by dividing the applied voltage by the measured current. An expected resistance is required to allow the measurement circuitry to adjust to the appropriate range.

Call RMeas(Resistance, Source, Measure, SrcVoltage, Limit, TestTime, Expected)
The RofRCMeas functional call uses a method that quickly charges the DUT until the capacitor is charged to the “ForcingV” voltage divided by 1000. This method maintains consistency with the CofRCMeas and RCMmeas functional calls. This call assumes that the value of the parallel capacitance being passed in the “ExpCap” parameter is valid. The current through the DUT is then measured, and the resistance is calculated as:

$$ R = \frac{V}{I} $$

Where:
- \( V \) = source voltage
- \( I \) = measured current

**Call RofRCMeas(Resistance, Source, Measure, ForcingV, Limit, ExpCap, ExpRes)**
RofRCMeas2

Call RofRCMeas2(Resistance, Source, Measure, PeakV, Limit, Frequency, ExpCap, ExpRes)

The RofRCMeas2 functional call uses a method that quickly charges the DUT up to the “PeakV” voltage and then the current is measured. This call assumes that the value of parallel capacitance passed in the “ExpCap” parameter is valid. The resistance is then calculated as:

\[ R = \frac{V}{I} \]

Where:
- \( V \) = source voltage
- \( I \) = measured current
RofRCMeas3

Call RofRCMeas3(Resistance, Source, Measure, PeakV, Limit, Frequency, ExpCap, ExpRes)

The RofRCMeas3 functional call uses a method that quickly charges the DUT up to the "PeakV" voltage and then the current is measured. This call assumes that the value of parallel capacitance passed in the "ExpCap" parameter is valid. The resistance is then calculated as:

\[
R = \frac{V}{I}
\]

Where:
- \( V \) = source voltage
- \( I \) = measured current
The ShortsMeas functional call executes a shorts test on a channel. Typically all of the channels will be guarded with GuardBd prior to executing any shorts tests. ShortsMeas will remove the guard on the given channel under test. A constant voltage is applied to the node for the test time, and the source output current is measured. The guard is restored to the source channel and the calculated resistance is returned where the user can compare it to an acceptable value.

Call ShortsMeas(Resistance, Channel, SrcVoltage, Limit, TestTime)
SHORTS TEST

This test checks all activated nets for net to net shorts. The test is performed by sequentially measuring the resistance between an individual net and all the other nets.

SHORTS TEST PARAMETERS

The conditions for the test are setup using the Shorts Test Parameter functions, as shown to the right.

**Source Voltage** - Value of the source voltage to be used during the test. Default value is 300 mV.

**Default Test Time** - Establishes the default test time for the test.

**Current Limit** - Establishes the default current limit for the test.

**Source Impedance** - Sets the source impedance limit for the test.

**Max Shorts Resistance** - Used to establish the resistance value below which a net short condition is said to exist. If the calculated Net impedance is greater than this value, this value is used instead.

**Min Opens Resistance** - Used to establish the resistance value above which a net open condition is said to exist. If the calculated Net impedance is less than this value, this value is used instead.

**Shorts Value %** - Percentage of calculated parallel resistance to be subtracted when setting the maximum resistance for a specific shorts test.

**Opens Value %** - Percentage of calculated parallel resistance to be added when setting the minimum resistance for a specific opens test.

**Min Shorts Threshold** - Calculated values below this default will disable the shorts test for the corresponding nets.

When the Generate Test Parameters command button is selected, AutoGen calculates the parallel test parameters and the sum of the capacitance for every individual net in the project with the Test Type designated as "Shorts&Opens". From this calculated value, AutoGen subtracts the "Shorts
Value %” to determine the maximum resistance for a short, and AutoGen adds “Opens Value %” to determine a minimum resistance for an open. If the maximum resistance for a short is greater than the default Max Shorts Resistance, the default is used during the shorts test. If the minimum resistance for an open is less than the default Min Opens Resistance, the default is used during the opens test. If the maximum resistance for a short is less than the Min Shorts Threshold, the shorts test is disabled. Typically, the resistance for the shorts and opens values should be between the shorts and opens thresholds. The test time is determined by the RC time constant based on the parallel resistance and the sum of the capacitance.

For nets with a Test Type designated as either just Shorts or just Opens, no calculations are performed, and the shorts and opens thresholds in the Net Editor will remain unchanged. For nets with high total capacitance, test times for these nets may be increased.

As mentioned above, when the maximum resistance for a short is calculated to be less than the Min Shorts Threshold, the shorts test is disabled. This prevents the program from designating two or more nets connected by a resistance less than the Min Shorts Threshold as shorted. Since the Shorts test and Opens test are automatically run before any parts test, this group of nets is skipped. To properly test these nets, the ShortsMeas functional call may be placed in a user test routine to determine the resistance. Each net in the group may be individually tested while the other nets in the group are manually unguarded.

The functional call parameters are covered on page xxx, and the measured resistance values for the group of nets are returned to the programmer in the Log Window.

As an example, nets 17, 29, and 42 are connected by 4Ω resistors. The resistance is less than the Min Shorts Threshold and the shorts test for these nets is disabled. Sample code for the User test for net 17 is shown on the next page. Since the OpenPinCheck for this net is of no significance, the condition flag is set to IPASS.

The OpenPinCheck functional call could be handled in the same manner.
Public Sub ShortsNET17(sd As ShortsData)
    Dim r As Double
    Unguard 29
    Unguard 42
    ShortsMeas r, 17, 0.100, 3, .001
    sd.Resistance = r
    sd.Condition = IIf(sd.Resistance > 10, IPASS, IFAIL)
    Guard 42
    Guard 29
End Sub

Public Sub OpensNET17(sd As ShortsData)
    sd.Condition = IPASS
End Sub
Small Resistance Measurement

The Small Resistance Measurement test is used to measure the resistance of small impedance components such as current sense resistors. This function sources a current through the DUT and measures it for reference. Then the function measures the voltage across the DUT through a third Matrix Relay channel which it uses to determine the effective impedance, in ohms. The DUT voltage must be supplied by one of the system Isolation Amplifiers, amplifying or buffering the differential voltage across the DUT. Any gain added by the Isolation Amplifier must be taken into account on the returned reading.

**Call RMeas2(Resistance, SourceCh, MeasureICh, MeasureVCh, ForcingI, Limit, TestTime, ExpRes)
The TPol functional call sources a voltage ramp into one side of the primary coil of a transformer and guards the other side. Then the finishing voltage of the induced voltage ramp is measured by connecting the ICAM RMUX circuitry to the secondary terminal that is in phase with the side of the primary being driven and guarding the other side of the secondary.

When the surrounding circuitry complies, it is possible to calculate the turns ratio by dividing the finishing source voltage by the measured voltage.

```
Call TPol(Voltage, PPlus, PMinus, SPlus, SMinus, RampV, Limit, TestTime, ExpVolts)
```
The TwoSrcDVM meas functional call measures the voltage drop across a forward-biased p-n junction. A constant current is applied to the source channel (anode) for the duration of the test time, and the cathode will be connected to the Guard bus. An external voltage is also routed through SMux bus 1 input for user-defined purposes such as gating an SCR. The returned voltage is measured at the source, and will equal the forward voltage drop of the p-n junction.

Call TwoSrcDVM meas(Voltage, Source, Measure, SrcCurrent, Limit, TestTime, Expected)

Caution: Guards should not be used with this functional call.
TwoSrcRMeas

Call TwoSrcRMeas(Resistance, Source, Measure, SrcVoltage, Limit, TestTime, Expected)

This call is functionally and parametrically identical to the RMeas functional call with one exception: The SMux bus 2 (the current source bus) is not cleared. This is an advantage when a specific test requires two sources, such as the closed contact resistance of a normally open relay. In this example, one source would be required to activate the coil and the other to provide the voltage sourced when measuring the contact resistance. The SMux call must be used prior to the TwoSrcRMeas call to route the second voltage source onto the ICAM board. SMux bus 1 is the only bus that can be used for this purpose.
The HFMux functional call routes one of eight high-frequency inputs to a measurement system input. The HF inputs are arranged in pairs, each of which goes to one of four range select circuits. Each range select circuit may be used by the ICAM board's Voltage Measurement System or by the Amplitude Measurement System (as SigA, Sig1, or Sig2 signals). Only one channel in each pair may be used at a time.

(The input path is disconnected if HFMux is followed by a later MMux or HFMux call to the same bus/range circuit).
The MMux functional call routes an ICAM channel to a measurement system input through one of four ICAM buses. The output of any range select circuit may be used by the ICAM board's Voltage Measurement system or by the Amplitude Measurement System (as SigA, Sig1, or Sig2).

(The signal path is disconnected if MMux is followed by a later SMux or HFMux call to the same bus/range circuit.)
The SMux functional call is used to provide an auxiliary source signal to any ICAM channel through one of the ICAM buses. Each of the four buses in the ICAM board may be disengaged from its default MDA output (or input) signal, and switched to one of its auxiliary sources. Then a connection will be made through the bus to the specified ICAM channel to provide the source signal to the fixture.
Fixturing

The UUT (Unit Under Test) is typically seated on a “Bed-of-Nails” fixture connected to the Testhead of the Series 2040 Test System. The ICAM subsystem employs Matrix Relay boards to make contact with all active nets on a UUT. Each net must be connected to a channel of a Matrix Relay board through the custom fixture. AutoGen software will automatically assign the Matrix Relay channels to these nets.

During construction of the fixtures, several steps and procedures must be followed:

1. Generating a Patchboard Map
2. Wiring the Matrix Relay boards
3. Wiring the Fixture ID
4. Wiring the Fixture Status
5. Wiring the Kelvin Matrix Relay Channels
6. Connecting the Fixture to Earth ground

Generating a Patchboard Map

Using the Windows Explorer, initiate the TRMAN program. If the operating system is Windows 95™, use the “Load Config from File” option from the File menu (shown to the right) to load the resource information into the TRMAN program. If the Operating system is Windows NT™ V4.0 or Windows XP™ Professional, use the “Read Config from Registry” option. If the project is being written on a test system other than the target system to be used in production, an appropriate configuration for the target system should be loaded. Consult the “Windows Programming Manual V1.0” (4200-0184) for more configuration options in TRMAN.

When the proper configuration is entered, use the Print option to “Print Map”. This option will generate a Patchboard map of the entire Patchboard. Before attempting any wiring, a copy of the map for the target system and an understanding of its contents is mandatory. Note that the pin definitions for the mnemonics used in the Patchboard map can also be printed from this menu.
Wiring the Matrix Relays

In the example, it shows to wire Matrix Relay channel #3 at Patchboard Pin #ZZ2 to the net called AMPINPUT_1_0062A. The nets that have a Matrix Relay channel assigned under the "Kel" column mean that these nets also need a kelvin channel wired to that net. For example, the pogo associated with net titled N00006_0062A will have two Matrix Relay channels wired to it. In this case channel numbers 5 and 258 at Patchboard Pins number ZZ3 and QQ2 will both be wired to the single pogo associated with net N00006_0062A. Documentation showing the device/net designation locations must be supplied by the manufacturer of the fixture or UUT. When the wiring is complete, it can be checked using the “Verify Fixture” dialog as described on page 40.

In addition to connecting a Matrix Relay channel to each of the nodes on the UUT, the four measurement/source ICAM patchboard pins need to be bussed to each of the Matrix Relay boards that are assigned to be used as measurement boards. See page 23 on how to assign Matrix Relay boards to be used as measurement boards. The four ICAM Patchboard pins are in the
left and right columns of rows 33 and 34. The four pins are used for the guard bus, the current source bus, the measurement bus, and the voltage source bus. In the patchboard map they are labeled as GrdB, CrSB, MsrB, and VtSB respectively. The connections are made as follows: The guard bus (left column row 33 on the ICAM) is bussed to the patchboard pin in the left column row 33 on each of the Matrix Relay boards assigned to be used as measurement boards. The current source bus (right column row 33 on the ICAM) is bussed to the patchboard pin in the right column row 33 on each of the Matrix Relay boards assigned to be used as measurement boards. The measurement bus (left column row 34 on the ICAM) is bussed to the patchboard pin in the left column row 34 on each of the Matrix Relay boards assigned to be used as measurement boards. And the voltage source bus (right column row 34 on the ICAM) is bussed to the patchboard pin in the right column row 34 on each of the Matrix Relay boards assigned to be used as measurement boards.

**Wiring the Fixture ID**

The ICAM board provides 16 patchboard pins to create the identification code for the test fixture. The 16 fixture ID pins are in the left and right columns of rows 23 through 30 on the ICAM board. They are used to represent a 16 bit hexadecimal number that identifies the test fixture. They are labeled as FID0 through FIDf on the patchboard map. See the patchboard map to get the bit number associated with each patchboard pin. The pin labeled FID0 is the lowest significant bit of the hexadecimal number and pin FIDf is the most significant bit. This hexadecimal number representing the fixtures ID can be read with the ‘FixID(‘) functional call. Any fixture ID pin connected to a digital ground patchboard pin on the ICAM board will be read as a 0. Fixture ID pins that are not connected to digital ground will be read as a 1. Therefore, to wire the fixture to have an ID of 0xFFEE, fixture ID bits #0 (FID0) and #4 (FID4) would be connected to digital ground and the rest would be left alone. The digital ground patchboard pins are labeled as DGND in the patchboard map.

**Wiring the Fixture Status**

The ICAM board provides two patchboard pins used to determine the status of up to two fixture vacuum wells. The status pin for the first well is in the left column of row 9 and the status pin for the second well is in the right column
of row 9. See the patchboard map to view their location. The ‘FixStat()’ functional call is used to read the current status of the vacuum wells. A fixture status pin needs to be brought low to signal the vacuum is engaged for that well. Please note that when a vacuum well is engaged (the pin is brought low), the ‘FixStat()’ functional call will return a 1 to signal that well is engaged (i.e. the functional call does not return the high/low state of the pin itself). A typical wiring method is to have an isolated pad on the UUT and have two pogo pins that will make contact with the pad when the UUT is engaged. One of the pogo pins is wired to the fixture status pin for that well and the other pin is wired to a digital ground patchboard pin on the ICAM board. Therefore, when the UUT is engaged and both pogo pins are in contact with the pad, the fixture status pin is connected to digital ground. The digital ground patchboard pins are labeled as DGND in the patchboard map.

If a switch of some sort is used to signal when the UUT is engaged, it must be wired such that digital ground is applied to the corresponding fixture status pin when that vacuum well is engaged.

Wiring the Kelvin Matrix Relay Channels

The source kelvin and measure kelvin patchboard pins on the ICAM board need to be wired to each of the Matrix Relay boards that have been assigned to be kelvin boards. See page 23 on how to assign Matrix Relay boards to be used as kelvin boards. The source kelvin and measure kelvin patchboard pins are in the left column of the ICAM board in rows 31 and 32 respectively. See the patchboard map to view their location. The source kelvin will be labeled as Skvn and the measure kelvin is labeled as Mkvn. The source kelvin pin on the ICAM needs to be bussed to the pin in the right column in row 33 on each of the kelvin Matrix Relay boards. The measure kelvin pin on the ICAM needs to be bussed to the pin in the left column in row 34 on each of the kelvin Matrix Relay boards.

Connecting the Fixture to Earth Ground

The fixture should be connected to earth ground to minimize the effects of 60 Hz noise during ICAM measurements. This can be accomplished by connecting one of the analog ground patchboard pins on the ICAM board to the fixture. An analog ground pin will be labeled as AGND in the patchboard map. The connection to the fixture must be placed at a location that also comes into contact with the tester’s earth ground. A common location to
connect to is the base of the fixture. The base of the fixture is the framework that holds the fixture’s Patchboard with all of the pins inserted into it. There usually is a screw that mounts the base to the fixture well. The connection to the fixture can be made by loosening the screw, wrapping the wire around it, and then tightening the screw again.
ICAM Calibration/Selftest
ICAM Calibration/Selftest

If the ICAM Calibration icon is selected from the Digalog program group, a runtime version of the Test Executive appears with a choice of the Calibrate or Functional test group.

Calibration

Before selecting between the ICAM calibration and functional tests, the reporting functions should be set up. From the File menu, select Configure, and the small “Report Utilities” dialog box shown to the right will appear. This box contains three separate ways of reporting the data collected during the tests. Data may be stored in a file, displayed to the screen, and/or printed. In addition, each of these three reporting functions can report “All Data,” “Fail Data,” or “No Data.” “All Data” includes all of the pass or fail sequences from the test. “Fail Data” includes only information on the tests that failed. “No Data” disables the reporting function. To select, merely use the mouse to check or uncheck each reporting function. If a reporting function is left unchecked, the options under that function appear “ghosted,” and that function is disabled. Once a reporting function is enabled, use the mouse to select the desired data masking function. For most situations, enable the “Window” and “Printer,” and select the “Fail Data” options on both.

When the selection of data reporting options is completed, click on the “OK” button to return to the Test Executive. From this screen, the user can select either the calibrate or functional test group option. The user now has a choice of executing the selected group (containing all of its tests) once, 10 times, 100 times, or continuously. The user can also click the Windows text tool on the
box to the left of the `<STOP>` button and enter the desired number of tests. Whatever is entered here will appear on the “button” directly above the box, and the specified number of tests will be executed when the button is selected.

**Configuration Summary** - This frame is used to display the report options selected from the Report Utilities dialog.

**Test Summary** - This frame is used to display test failures. The cycle number, Menu name, Program name, and number of total failures are displayed.

If the user double-clicks on the calibrate test group, the following menu is displayed listing the various calibration routines for the ICAM hardware.

The user may select an individual test and run it once, 10 times, 100 times, or continuously as before. The user can also click the Windows text tool on the box to the left of the `<STOP>` button and enter the desired number of tests. Whatever is entered here will appear on the “button” directly above the box, and the specified number of tests will be executed when the button is selected. In addition, the user can execute all of the individual tests by checking the “ALL” box with the mouse and then selecting the desired number of iterations. If two or three adjacent tests are to be run, they may be selected.
using only the mouse. For example, if test #3 and test #4 need to be run, place the mouse pointer on test #3 and hold the left button down. Drag the pointer down to test #4, and both tests will be selected.

If a random series of tests is desired, place the mouse pointer on the box under “Run Sequence.” When the Windows text tool appears, click the left mouse button. At the blinking cursor, any sequence of tests can be entered.

The test numbers, however, must follow this sequence:

- ‘E’ (Execute) signifies the beginning of a sequence or acts as a separator between sequences.
- ‘:’ (Colon) signifies that the following number indicates the amount of iterations to perform of the previous sequence.
- ‘,’ (Comma) shares the definition of ‘E’ as a separator between sequences.
- ‘-’ (Dash) signifies inclusive or ‘through’.

**EXAMPLE:**  **E1-3:5,6-9:2,1-5**

This sequence executes tests 1 through 3 five times, tests 6 through 9 twice, and tests 1 through 5 once.

While viewing the Test Executive menu displaying the “calibrate” and “functional” test groups, click on “calibrate” group to select it, and click on the “1” button to run the entire calibration group once. If any failures occur during this routine, a hardware problem is most likely the cause. A list of the specific calibration programs and probable causes is shown below and on the next few pages.

**CalADInput**

This program calibrates the path from the A/D multiplexer to the A/D converter. The A/D multiplexer is a 16:1 mux that is used to select the signal to be measured by the A/D converter. Thirty samples are taken and a “least squares” statistical analysis is performed to obtain the gain and offset values for this path. The TDAC in the Turbo Selftest Assembly is used as a source. This calibration program result is used by other calibration programs and must be performed first. If this calibration fails, the gain and offset values are not stored, and the previous values in the calibration constants file are retained. The Turbo Selftest Assembly is required for this test.
• If any failures occur during this calibration, the problem is most likely on the ICAM board.

• Since the TDAC in the Turbo Selftest Assembly required for the test, it could also cause the failures.

• Power to the Testhead and connections should always be checked.

CalArbISource
This program calibrates the 16-bit ARB when it is used as a current source. Thirty samples are taken and a “least squares” statistical analysis is performed to obtain the gain and offset values for the ARB. If the calibration fails, the gain and offset values are not saved, and the previous values in the calibration constants file are retained. The Turbo Selftest Assembly is required for this test.

• If any failures occur during this calibration, the problem is most likely on the ICAM board.

• Since the Turbo Selftest Assembly is required for the test, it could also cause the failures.

• If all of the current calibrations (CalArbISource, CalDCISrc, and CalIRail) fail, the problem is most likely in the 40 Volt supply since it is used when sourcing current for all three tests.

• Power to the Testhead and connections should always be checked.

CalArbVSource
This program calibrates the 16-bit ARB when it is used to source voltage. Thirty samples are taken and a “least squares” statistical analysis is performed to obtain the gain and offset values for the ARB. If the calibration fails, the gain and offset values are not saved, and the previous values in the calibration constants file are retained. The Turbo Selftest Assembly is not used in this calibration.

• If any failures occur during this calibration, the problem is most likely on the ICAM board.

• Power to the Testhead and connections should always be checked.

CalDCISrc
This program calibrates the secondary DC 12-bit current source. Thirty samples are taken and a “least squares” statistical analysis is performed to
obtain the gain and offset values for the 12-bit D/A. If the calibration fails, the gain and offset values are not saved, and the previous values are retained. The Turbo Selftest Assembly is required for this calibration.

- **If any failures occur during this calibration, the problem is most likely on the ICAM board.**
- **Since the Turbo Selftest Assembly is required for this test, it could also cause the failures.**
- **If all of the current calibrations (CalArbISource, CalDCISrc, and CalIRail) fail, the problem is most likely in the 40 Volt supply since it is used when sourcing current for all three tests.**
- **Power to the Testhead and connections should always be checked.**

**CalIMon**

This program calibrates the IMON resistor on the ICAM board. The IMON resistor can be used to measure the current coming out of the 16 bit ARB when its sourcing a voltage. The calibration is done by programming an output voltage from the ARB. This voltage comes out to a Patchboard pin and to the Turbo Selftest. A resistor in the Turbo Selftest routes this voltage to ground. The voltage at the high side of the resistor is then measured. The resistor in the Selftest Assembly has previously been measured so its exact value is known. The measured voltage divided by the measured resistance gives the exact current through this resistor. This current will be the same as the current through the IMON resistor. The voltage across the IMON resistor is then measured as a bit count. The bit count is then calibrated to the current measured through the Selftest Assembly resistor. Thirty samples are taken and a “Least Squares” is performed to obtain a gain and offset value. This calibration needs a Turbo Selftest Assembly with an ICAM selftest card in the same slot as the ICAM Testhead board.

- **The ICAM Selftest board needs to be calibrated.**
- **If any failures occur during this calibration, the problem is most likely on the ICAM board.**
- **The VO UT Patchboard pin must make contact with the corresponding Selftest Assembly pin.**
- **Power to the Testhead and connections should always be checked.**
CalIRail

This program calibrates the circuitry that limits the output current when either the 16-bit ARB or the secondary 12-bit source DAC is used to source current. Thirty samples are taken and a “least squares” statistical analysis is taken to obtain the gain and offset values. If the calibration fails, the gain and offset values are not saved, and the previous values in the calibration constants file are retained. The Turbo Selftest Assembly is required for this test.

- If any failures occur during this calibration, the problem is most likely on the ICAM board.
- Since the Turbo Selftest Assembly is required, it could also be the problem.
- If all of the current calibrations fail, (CalArbISource, CalDCISrc, and CalIRail) the problem is most likely in the 40 Volt supply since it is used when sourcing current for all three tests.
- Power to the Testhead and connections should always be checked.

CalMeasAmp

This program calibrates all six ranges on the measurement amplifier. The measurement amplifier is used to test resistors, capacitors, and diode voltage drops. Thirty samples are taken and a “least squares” statistical analysis is performed to obtain the gain and offset values for each range. If any one of these calibrations fails, none of the values are stored, and the previous values in the calibration constants file are retained. The Turbo Selftest Assembly is required for this program.

- If any failures occur during this calibration, the problem is most likely on the ICAM board.
- Since the Turbo Selftest Assembly is required for this test, it could also cause the failures.
- Power to the Testhead and connections should always be checked.

CalMeasKelvin

This program calibrates the measurement circuitry when a measurement kelvin channel is used. The calibration is very similar to the CalMeasAmp program except only the 100 and 10 mAmp ranges are calibrated. Only these two ranges are calibrated because kelvin measurement method is not used for
the lower four current measurement ranges. There is a minor change to the measurement circuitry to allow for accurate kelvin measurements so the calibration has been separated from the CalMeasAmp program. The calibration is performed by applying a known voltage (from TDAC) to the kelvin measurement circuit. The bit count associated with this voltage is then measured. Thirty samples are taken and a “Least Squares” is performed to obtain a gain and offset value for each range. This calibration needs a Turbo Selftest unit with an ICAM Selftest card in the same slot as the ICAM Testhead board.

- The ICAM Selftest board needs to be calibrated.
- If a failure occurs, the ICAM board is most likely faulty.
- Power to the Testhead and connections should always be checked.

CalSig3

This program calibrates the Sig3 path to the ICAM measurement circuitry. The Sig3 signal comes onto the ICAM board from the Testhead’s analog motherboard. TDAC provides the known voltage to the Sig3 line. Thirty samples are taken and a “Least Squares” is performed to obtain a gain and offset value for each range. This calibration needs a Turbo Selftest Assembly with an ICAM Selftest card in the same slot as the ICAM Testhead board. The Sig3 path is used to measure the Testhead and Patchboard Power Supplies in the IPower_Mon_f Selftest program.

- If a failure occurs, the ICAM board is most likely faulty.
- Since the TDAC is used, the Turbo Selftest Assembly could also cause a problem.
- Power to the Testhead and connections should always be checked.

CalVMeas

This program calibrates the ranging measurement circuitry on the ICAM board. The ranging measurement circuitry is used to route a voltage from the Patchboard to the ICAM’s measurement circuit. The ranging circuitry can be used to divide the input voltage from the Patchboard by a factor of 100 or 10, multiply by a factor of 10, or pass the full voltage through. There is a gain and offset value for each of the four ranges on each of the four groups (a total of 16 gain and offset values). The Turbo Selftest TDAC is used to supply the voltage. Thirty samples are taken and a “Least Squares” is used to obtain each
gain and offset value. This calibration needs a Turbo Selftest Assembly with an ICAM Selftest card in the same slot as the ICAM Testhead board.

- If a failure occurs, the ICAM board is most likely faulty.
- Since the TDAC is used, the Turbo Selftest Assembly could also cause a problem.
- The High Frequency Mux & SMux Patchboard pins must make contact with the Selftest pins.
- Power to the Testhead and connections should always be checked.

**CalVMon**

This program calibrates the VM on signal. This calibration needs to be done to be able to calibrate the current source circuitry (both the 12 bit DC source and the 16 bit ARB). The calibration is done by applying a DC current to an open load. The current source rails are then varied and the voltage at the output of the current source is measured using the ranging circuitry calibrated in CalVM eas. The bit counts are then measured at VM on and this bit count is calibrated to the voltage previously measured. Thirty samples are taken and a “Least Squares” is performed to obtain a gain and offset value. This calibration needs a Turbo Selftest Assembly with an ICAM Selftest card in the same slot as the ICAM Testhead board.

- If a failure occurs, the ICAM board is most likely faulty.
- The CalVM eas calibration needs to pass.
- If the current source calibrations also fail, the problem could be the 40 volt supply since this supply is used to set the current source rails.
- Power to the Testhead and connections should always be checked.

**CalVRef**

This program calibrates the VRef signal. This calibration needs to be done to be able to calibrate the current source circuitry (both the 12 bit DC source and the 16 bit ARB). The calibration is done by applying a DC current to an open load. The current source rails are then varied and the voltage at the output of the current source is measured using the ranging circuitry calibrated in CalVM eas. The bit counts are then measured at VRef and this bit count is calibrated to the voltage previously measured. Thirty samples are taken and a
“Least Squares” is performed to obtain a gain and offset value. This calibration needs a Turbo Selftest Assembly with an ICAM Selftest card in the same slot as the ICAM Testhead board.

- If a failure occurs, the ICAM board is most likely faulty.
- The CalVMeas calibration needs to pass.
- If the current source calibrations also fail, the problem could be the 40 volt supply since this supply is used to set the current source rails.
- Power to the Testhead and connections should always be checked.

Selftest

From the initial Test Executive screen, click on the “functional” test option to select it, and click on the “1” button to run the entire functional routine once. If failures occur during this routine after the ICAM system has already passed a calibration, check these common problems:

**TURBO Selftest Assembly Improperly Installed** - Check to make sure that the assembly is properly installed on the Patchboard. Also, check the Turbo Selftest Assembly cable for damage and loose connections.

**UUT Power Supplies** - Make sure that the UUT Power Supplies are on and functioning.

**Testhead Power Supply Fault** - Check the Testhead Power Supply Indicator LEDs. If any of the LEDs are not lit, cycle the Testhead power switch off and on, and recheck the LEDs. If the “external fault” LED is not lit, test all power supplies and recheck wiring and connections.

After the above common problems have been eliminated, a failure in the ICAM Functional Selftest indicates a hardware problem. A list of the specific functional programs and their probable failures is shown below and in the following pages.

**ADMeas_f**

The ADMeas_f Functional Selftest program tests the full input range of the analog-to-digital convertor (ADC) on the ICAM board using its test digital-to-analog convertor (TDAC) input channel. The TDAC’s output range is tested from -4V to +4V in 0.2V increments. The Turbo Selftest Assembly is required.
for this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it initializes the ADC to take readings of its TDAC input channel. The TDAC is programmed to -4.0V before triggering the ADC. The ADC’s reading is then retrieved and compared against the minimum and maximum limits. The TDAC is programmed to the next test value, and the trigger-measurement-comparison process is repeated until the entire input range of the ADC has been tested.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Since the Turbo Selftest Assembly is required for this test, it could also cause the failure(s).**
- **Power to the Testhead and connections should always be checked.**

**ADMem_f**

The ADMem_f Functional Selftest program tests the memory that the analog-to-digital convertor (ADC) on the ICAM board uses to store its readings. It tests the address and data line interfaces as well as all of the locations in the memory chips. The Turbo Selftest Assembly is not used in this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then tests for short-circuited and open-circuited data and address lines. Next, it tests each and every address location by first “marching” a logic 1 followed by a logic 0. Finally, the program tests the hardware that controls the address lines to verify that all of the memory locations are accessible.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Power to the Testhead and connections should always be checked.**

**ArbBrst_f**

The ArbBrst_f Functional Selftest program tests the burst mode of the ARB circuitry on the ICAM board. The complete range of bursts from 1 to 255 is verified. The Turbo Selftest Assembly is not used in this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then
initializes the ARB to toggle its MPULSE signal once per cycle and sets up the Time Measurement Circuit (TMC) on the ICAM board to record the number of pulses on its MPULSE input channel within a 100-mS time interval. Next, the ARB’s burst counter is configured for 1 pulse before triggering the ARB and the TMC. The TMC’s reading is then retrieved and compared to the expected value. The burst counter is incremented to the next test value, and the trigger-measurement-comparison process is repeated until all 255 burst counter settings are tested.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Power to the Testhead and connections should always be checked.**

**ArbFreq_f**

The ArbFreq_f Functional Selftest program tests the frequency accuracy of the ARB’s frequency generator on the ICAM board. The test frequencies range from 10 Hz to 10 MHz incrementing on a per-decade basis. The Turbo Selftest Assembly is not used in this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then initializes the TMC to record the number of pulses on its MCLK input channel within either a 10mS or 100mS time interval. Next, the ARB’s frequency generator is programmed to a value of 10 Hz before triggering the ARB and the TMC. The TMC’s reading is then retrieved and compared against the minimum and maximum limits. The ARB’s frequency generator is reprogrammed to the next incremental (by decade) value, and the trigger-measurement-comparison process is repeated until all of the frequency settings have been tested.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Power to the Testhead and connections should always be checked.**

**Arb_I_f**

The Arb_I_f Functional Selftest program tests the 16-bit ARB circuitry used to generate the 5mA and 50mA bipolar current ranges for the current source on the ICAM board. The 5mA range is tested from -4.5μA to -4.5mA and from +4.5μA to +4.5mA. The 50mA range is tested from -5mA to -50mA and from +5mA to +50mA. The Turbo Selftest Assembly is required for this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then
verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it enables the load resistor on the Turbo Selftest Assembly for the current source’s 5mA range. It also sets up the analog-to-digital convertor (ADC) to read the voltage drop across the current source’s sense resistor. Then, it programs the ARB to source 4.5uA before triggering the ARB and the ADC. The ADC’s reading is then retrieved and compared to the minimum and maximum limits. The ARB is programmed to the next incremental value, and the trigger-measurement-comparison process is repeated until all of the positive and negative 5mA and 50mA current ranges of the ARB are tested.

- If any failures occur, the problem is most likely on the ICAM board.
- Since the Turbo Selftest Assembly is required for this test, it could also cause the failure(s).
- Power to the Testhead and connections should always be checked.

**Arb_V_f**

The Arb_V_f Functional Selftest program tests the 16-bit ARB’s output amplifier circuitry and the VOUT signal path to the ICAM measurement circuitry. The ARB is programmed to output a range of voltages from its maximum value to its minimum value. This voltage is then measured using the VOUT signal path to the A/D Converter. The test is performed for both a normal ARB voltage source and a Kelvin voltage source.

- If any failures occur, the problem is most likely on the ICAM board.
- Power to the Testhead and connections should always be checked.

**ArbILim_f**

The ArbILim_f Functional Selftest program tests the current-limiting protection circuitry for the ARB on the ICAM board. It checks both the 200mA and 1A current ranges. This program verifies that the current-limiting protection circuitry is either enabled or disabled under the appropriate circumstances. The Turbo Selftest Assembly is required for this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it enables the load resistor on the Turbo Selftest Assembly for the ARB. It also sets up the analog-to-digital convertor (ADC) to read the ARB’s current output.
Then, it enables the ARB’s 200mA current range and programs the ARB to source +500mA before triggering the ARB and the ADC. The ADC’s reading is then retrieved and compared to the minimum and maximum limits of 200mA and 275mA, respectively. Next, the ARB is programmed to source +175mA, and the trigger-measurement-comparison process is repeated. The ADC’s reading is now compared to the nominal value of (175mA multiplied by the load resistance) +/-5%. This is then repeated for the current sinking values of 175mA and 500mA. Next, the ARB’s 1A current range is enabled and the ARB’s current-sourcing and current-sinking capabilities are tested at the settings of 500mA and 750mA.

- If any failures occur, the problem is most likely on the ICAM board.
- Since the Turbo Selftest Assembly is required for this test, it could also cause the failure(s). It may only need to be recalibrated.
- Power to the Testhead and connections should always be checked.

**ArbIMon_f**

The ArbIMon_f Functional Selftest program tests the ARB’s current output measurement circuitry. The ARB’s 1A current range is tested from -900mA to +900mA in 60mA increments. The Turbo Selftest Assembly is required for this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it enables the load resistor on the Turbo Selftest Assembly for the ARB. It also sets up the analog-to-digital convertor (ADC) to read the ARB’s current output. Then, it enables the ARB’s 1A current range and programs the ARB to deliver +900mA before triggering the ARB and the ADC. Next, the ADC’s reading is retrieved and compared to the minimum and maximum limits. The ARB is then programmed to the next incremental value, and the trigger-measurement-comparison process is repeated until the entire range is tested.

- If any failures occur, the problem is most likely on the ICAM board.
- Since the Turbo Selftest Assembly is required for this test, it could also cause the failure(s). It may only need to be recalibrated.
- Power to the Testhead and connections should always be checked.
ArbMem_f

The ArbMem_f Functional Selftest program tests the memory that the ARB on the ICAM board uses to retrieve its wave forms. It tests the address and data line interfaces as well as all of the locations in the memory chips. The Turbo Selftest Assembly is not used in this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then tests for short-circuited and open-circuited data and address lines. Finally, it tests each and every address location by first “marching” a logic 1 followed by a logic 0.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Power to the Testhead and connections should always be checked.**

Arly_Dig_f

The Arly_Dig_f Functional Selftest program tests the digital circuitry on the Auxiliary Relay, Auxiliary FET, Power Relay, and High Current FET boards in the Testhead. The Turbo Selftest Assembly is not used in this functional selftest program.

If multiple boards or a combination of boards exists in the system, the program tests each one in the order indicated by its dip switch settings i.e., 0, 1, 2, 3.

- **If any failures occur, the problem is most likely on the indicated board.**
- **Power to the Testhead and connections should always be checked.**

DCI_f

The DCI_f Functional Selftest program tests the secondary 12 bit digital-to-analog convertor (DAC) used to generate the 5mA and 50mA unipolar current ranges for the current source on the ICAM board. The 5mA range is tested from +45uA to +4.5mA. The 50mA range is tested from +5mA to +50mA. The Turbo Selftest Assembly is required for this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it enables the load resistor on the Turbo Selftest Assembly for the current source’s 5mA range. It also sets up the analog-to-digital convertor (ADC) to...
read the voltage drop across the current source’s sense resistor. Then, it programs the secondary 12-bit DAC to source 45uA before triggering the ARB and the ADC. The ADC’s reading is then retrieved and compared to the minimum and maximum limits. The DAC is programmed to the next incremental value, and the trigger-measurement-comparison process is repeated until the 5mA and 50mA current ranges are fully tested.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Since the Turbo Selftest Assembly is required for this test, it could also cause the failure(s).**
- **Power to the Testhead and connections should always be checked.**

**DelayC_f**

The DelayC_f Functional Selftest program tests the delay counter for the analog-to-digital convertor (ADC) on the ICAM board. Due to time constraints, only 16 tests are performed to verify the integrity of the delay counter hardware. The Turbo Selftest Assembly is not used in this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. Then, it sets the identical frequency for the sample clock and for the ARB. Next, it initializes the ADC’s delay counter to the first test delay value. It also programs the ARB to generate a single, unit-long pulse at the same moment that the delay counter will time out and cause the analog-to-digital convertor (ADC) to measure its input. After the ARB and the ADC have been triggered, the ADC’s reading of the ARB’s output is retrieved and compared to the expected value of 2.5V +/-5%. The delay counter and the ARB are then programmed with the next test value, and the trigger-measurement-comparison process is repeated until all of the test delay values have been checked.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Power to the Testhead and connections should always be checked.**

**Event_f**

The Event_f Functional Selftest program tests the five fixed-frequency clock sources of the ADC’s delay counter and the five time interval settings for the Time Measurement Circuit (TMC) on the ICAM board. The Turbo Selftest Assembly is not used in this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. Next,
it routes the first fixed-frequency clock source for the ADC’s delay counter to the TMC, which is then set up to use its first time interval setting. After triggering, the TMC’s pulse count is retrieved and compared against minimum and maximum limits. The next clock source is selected, and the trigger-measurement-comparison process is repeated until all of the possible clock source/time interval combinations have been tested.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Power to the Testhead and connections should always be checked.**

**Fixture_f**

The Fixture_f Functional Selftest program tests the digital circuitry on the ICAM board that interfaces with the fixture. This includes the status lines for the left and right vacuum wells, the fixture ID interface, and the four open-collector driver outputs. The Turbo Selftest Assembly is required for this functional selftest program.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Since the Turbo Selftest Assembly is required for this test, it could also cause the failure(s).**
- **Power to the Testhead and connections should always be checked.**

**IArly_f**

The IArly_f Functional Selftest program tests the relays (or switches) on all of the Auxiliary Relay, Auxiliary FET, and Power Relay boards in the system. It is assumed that the measurement system is working although it doesn’t need to be calibrated. The Turbo Selftest Assembly is required for this functional selftest program.

The program begins by first searching the Testhead for the Amplitude Measurement System (AMS) board. If the AMS board is present, a separate procedure called nArly_f is executed to test these relay board(s) using the AMS and Relay Multiplexer boards along with the Turbo Selftest Assembly.

Otherwise, the program searches the Testhead for the presence of the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it searches for all existing Auxiliary Relay, Auxiliary FET, and Power Relay board and verifies that each board has a corresponding Selftest board.
The reference voltage (generated by the high-precision test digital-to-analog convertor - TDAC, in the Turbo Selftest Assembly) is then applied to the first relay (or switch) on the first board through a resistor. A second, matched resistor connects the other end of the relay (or switch) to ground. The analog-to-digital convertor (ADC) on the ICAM board should measure between 45-55% of the reference voltage across the relay (or switch) when it is closed. The ADC should measure less than 5% of the reference voltage when the relay (or switch) is open.

Then, the rest of the relays (or switches) on the board are checked. If multiple boards or a combination of boards exists in the system, the program tests each one in the order indicated by its dip switch settings i.e., 0, 1, 2, 3.

- **If any failures occur, the problem is most likely on the indicated relay board.**
- **If the AMS board is present, it and/or the Relay Multiplexer board could also cause the failure(s).**
- **If the AMS board is not present, the ICAM board could also cause the failure(s).**
- **Since the Turbo Selftest Assembly is required for this test, it too could cause the failure(s).**
- **Power to the Testhead and connections should always be checked.**

**IC_dig_f**

- This program tests the digital circuitry on the ICAM board. It writes to and reads back from each register. It reports a failure each time that the received data does not match the written data. The Turbo Selftest Assembly is not used in this program.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Power to the Testhead and connections should always be checked.**

**IMrly_f**

The IMrly_f Functional Selftest program tests the relays on all of the Matrix Relay boards in the system. It is assumed that the measurement system is working although it doesn’t need to be calibrated. The Turbo Selftest Assembly is required for this functional selftest program.
The program begins by first searching the Testhead for the Amplitude Measurement System (AMS) board. If the AMS board is present, a separate procedure called nMry_f is executed to test the Matrix Relay board(s) using the AMS and Relay Multiplexer boards and the Turbo Selftest Assembly.

Otherwise, the program searches the Testhead for the presence of the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it searches for all existing Matrix Relay boards and verifies that each board has a corresponding Selftest board.

The reference voltage (generated by the high-precision test digital-to-analog convertor - TDAC, in the Turbo Selftest Assembly) is then applied to the first relay on the first board through a resistor. A second, matched resistor connects the other end of the relay to ground. The analog-to-digital convertor (ADC) on the ICAM board should measure between 45-55% of the reference voltage across the relay when it is closed. The ADC should measure less than 5% of the reference voltage when the relay is open.

Then, the rest of the relays on the board are checked. If multiple boards exist in the system, the program tests each one in the order indicated by its dip switch settings i.e., 0, 1, 2, 3.

- If any failures occur, the problem is most likely on the indicated Matrix Relay board.
- If the AMS board is present, it and/or the Relay Multiplexer board could also cause the failure(s).
If the AMS board is not present, the ICAM board could also cause the failure(s).

Since the Turbo Selftest Assembly is required for this test, it too could cause the failure(s).

Power to the Testhead and connections should always be checked.

IPower_Mon_f

The IPower_Mon_f Functional Selftest program checks the DC voltage outputs of all of the internal Testhead and external Patchboard power supplies. The Testhead board in the first slot must contain a high-voltage test multiplexer (TMUX). The measurement system must have been successfully calibrated. Also, the Turbo Selftest Assembly is required for this functional selftest program.

The program begins by first searching the Testhead for the Amplitude Measurement System (AMS) board. If the AMS board is present, a separate procedure called Power_Mon_f is executed to test the power supplies using the Turbo Selftest Assembly.

Otherwise, the program searches the Testhead for the presence of the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it routes the output of each power supply through the high-voltage TMUX on the first Testhead board to the SIG3 input channel of the analog-to-digital convertor (ADC) on the ICAM board. A measurement is taken and then compared against the corresponding minimum and maximum limits. The path of the test digital-to-analog convertor (TDAC) in the Turbo Selftest Assembly through the TMUX to the ADC is also tested.

If any failures occur, the problem is most likely in the indicated power supply or with its wiring.

If the AMS board is present, it and/or the Relay Multiplexer board could also cause the failure(s).

If the AMS board is not present, the ICAM board could also cause the failure(s).

Since the Turbo Selftest Assembly is required for this test, it too could cause the failure(s).
If multiple power supplies exhibit failures, check the Testhead Power Supply Controller board and the high-voltage TMUX on the Testhead board in the first slot.

Power to the Testhead and connections should always be checked.

**IRail_f**

The IRail_f Functional Selftest program tests the amplifier circuitry of the “RAILDAC” digital-to-analog convertor (DAC) which provides the adjustable power supply voltage for the current source on the ICAM board. The DAC’s output is tested from 3V to 35V in 1.1V increments. The Turbo Selftest Assembly is required for this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it sets up the analog-to-digital convertor (ADC) to read the voltage output of the current source, VMON. Then, it programs the IRAIL DAC to 3.0V before triggering the ARB and the ADC. Next, the ADC’s reading is retrieved and compared to the minimum and maximum limits. The IRAIL DAC is then programmed to the next incremental value, and the trigger-measurement-comparison process is repeated until the entire output range has been tested.

- If any failures occur, the problem is most likely on the ICAM board.
- Since the Turbo Selftest Assembly is required for this test, it could also cause the failure(s).
- Power to the Testhead and connections should always be checked.

**IRDACs_f**

The IRDACs_f Functional Selftest program tests the “IDAC” and “RAILDAC” digital-to-analogue convertors (DACs) on the ICAM board. Each DAC’s output is tested from 0V to 40% of full-scale output. The Turbo Selftest Assembly is not used in this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. Next, it sets up the analog-to-digital convertor (ADC) to read its “RAILDAC” DAC’s input channel. Then, it programs the “RAILDAC” DAC to -10V before triggering the ARB and the ADC. Next, the ADC’s reading is retrieved and compared to the minimum and maximum limits. The “RAILDAC” DAC is programmed to the next incremental value, and the trigger-measurement-comparison process is repeated until the entire output range has been tested.
comparison process is repeated until the test range has been covered. Then, the ADC is set up to read its “IDAC” DAC’s input channel, and the “IDAC” DAC is tested in the same manner as its counterpart.

- If any failures occur, the problem is most likely on the ICAM board.
- Power to the Testhead and connections should always be checked.

**Meas_F_f**

The Meas_F_f Functional Selftest program tests the input frequency range of the Time Measurement Circuit (TMC) on the ICAM board. The test frequencies range from 10 Hz to 3 MHz incrementing on a per-decade basis. The Turbo Selftest Assembly is not used in this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then initializes the TMC to record the number of pulses on its MCLK input channel within either a 10mS or 100mS time interval. Next, the ARB’s frequency generator is programmed to a value of 10 Hz before triggering the ARB and the TMC. The TMC’s reading is then retrieved and compared against the minimum and maximum limits. The ARB’s frequency generator is reprogrammed to the next incremental (by decade) value, and the trigger-measurement-comparison process is repeated until all of the frequency settings have been tested.

- If any failures occur, the problem is most likely on the ICAM board.
- Power to the Testhead and connections should always be checked.

**Meas_I_f**

The Meas_I_f Functional Selftest program tests the current measurement amplifier on the ICAM board. It checks all six current ranges (100mA, 10mA, 1mA, 100uA, 10uA, & 1uA) and their optional Kelvin-measurement connections. The Turbo Selftest Assembly is required for this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, the analog-to-digital convertor (ADC) is set up to read the output of the current measurement amplifier, VRM EAS. Then, the program selects the first current range of the current measurement amplifier and its corresponding input resistor on the Turbo Selftest Assembly. Next, it programs the test digital-to-
analog convertor (TDAC) on the Turbo Selftest Assembly to -8.7V before triggering the ARB and the ADC. Then, the ADC’s reading is retrieved and compared to the minimum and maximum limits. The TDAC is programmed to the next incremental value, and the trigger-measurement-comparison process is repeated for the full range of TDAC’s output and for the remaining five current ranges.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Since the Turbo Selftest Assembly is required for this test, it could also cause the failure(s).**
- **Power to the Testhead and connections should always be checked.**

### Meas_V_f

The Meas_V_f Functional Selftest program tests the four voltage ranges (0.2V, 2V, 20V, & 200V) of each of the four voltage input relay multiplexer (RMUX) groups on the ICAM board. Each group provides a separate input signal to the analog-to-digital convertor (ADC). The Turbo Selftest Assembly is required for this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it configures the first RMUX to its 200V range. Then, it routes the output of the test digital-to-analog convertor (TDAC) on the Turbo Selftest Assembly to the first RMUX group. Next, the program sets up the ADC to read the output of the first RMUX group, VMEAS0. Then, it programs the TDAC to -9.375 before triggering the ADC. Next, the ADC’s reading is retrieved and compared to the minimum and maximum limits. The TDAC is then programmed to the next incremental value, and the trigger-measurement-comparison process is repeated until the entire 200V range has been tested. Then, the 20V, 2V, and 0.2V ranges are tested in order using the initial TDAC setting of 9.375V, 1.75V, and 0.175V, respectively. Afterwards, the other three RMUX groups are tested in a similar fashion.

- **If any failures occur, the problem is most likely on the ICAM board.**
- **Since the Turbo Selftest Assembly is required for this test, it could also cause the failure(s).**
- **Power to the Testhead and connections should always be checked.**
MRLY_dig_f

This program tests all of the I/O bits that can be read back on the all of the Matrix Relay boards in the system. It also tests the image of the 256 relays in the controller memory, the relay drivers, and the readback serial shift registers on each board.

The second part of the test is accomplished by changing one bit at a time in the image and downloading the image through the driver shift registers. The driver outputs are read back through the readback serial shift registers. Finally, the received values are compared to the set bits, and any failures are recorded. A Turbo Selftest Assembly is required for this test.

- If any failures occur, the problem is most likely on the indicated Matrix Relay board.
- Since the Turbo Selftest Assembly is required for the test, it could also cause the failure(s).
- Power to the Testhead and connections should always be checked.

SampleC_f

The SampleC_f Functional Selftest program tests the sample counter of the analog-to-digital convertor (ADC) on the ICAM board. The entire sampling range from 1 to 32767 is tested using the ARB as the input source. The Turbo Selftest Assembly is not used in this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then programs the ARB to a non-zero DC output and routes the signal to the ADC. For each test sample value, the ADC’s sample memory is first reset to zero, the ADC’s sample counter is programmed with the test sample value, and both the ARB and the ADC are triggered. Once the ADC has completed taking its readings, its memory is checked to verify it contains only the specified number of readings.

- If any failures occur, the problem is most likely on the ICAM board.
- Power to the Testhead and connections should always be checked.

SHFMux_f

The SHFMux_f Functional Selftest program tests all of the possible input signal paths (and their relays) involving the four voltage relay multiplexer (RMUX) groups (#0-3) and the current measurement amplifier. Each group has four
source bus input channels (#0-3), two high-frequency input channels (#0-1), and a default source bus input source (except for Group #0). The current measurement amplifier can be connected to any one of the twenty-three (23) possible input signals. The Turbo Selftest Assembly is required for this functional selftest program.

The program begins by first searching the Testhead for the ICAM board. It then verifies that the Turbo Selftest Assembly is present and that the assembly contains an ICAM Selftest board in the same slot as the ICAM board. Next, it programs the test digital-to-analog converter (TDAC) on the Turbo Selftest Assembly, the ARB, and the secondary 12-bit current source DAC to provide the test signals that will be routed through the various signal paths and relays. Then, the program configures the first RMUX to select its four source bus input channels and two high-frequency input channels one at a time in order whereby the analog-to-digital converter (ADC) is triggered for each configuration. The ADC’s reading is retrieved and compared to the minimum and maximum limits. Next, the input paths and relays of the remaining three groups are tested in a similar manner. Also, the output of the secondary 12-bit current source DAC is routed to the current measurement amplifier and tested to verify its input paths.

- If any failures occur, the problem is most likely on the ICAM board.
- Since the Turbo Selftest Assembly is required for this test, it could also cause the failure(s).
- Power to the Testhead and connections should always be checked.

**XOPhase_f**

The XOPhase_f Functional Selftest program tests the crossover detectors on the ICAM board. This circuit can be used in conjunction with the Time Measurement Circuit (TMC) to measure the frequency of the ARB’s output waveform. The Turbo Selftest Assembly is not used in this functional selftest program.

- If any failures occur, the problem is most likely on the ICAM board.
- Power to the Testhead and connections should always be checked.
Wirelist Import Format
**Appendix A - AutoGen Import Description Rev. 0.00**

**AutoGen Wirelist Importing**

This file describes what AutoGen is looking for in the wirelist. There are primarily two sections that are searched:

- Component List
- Wire List

The Component List section begins with a line:

&lt;&lt; Component List &gt;&gt;

The Wire List section begins with a line:

&lt;&lt; Wire List &gt;&gt;

The Component List section contains information about specific parts. The Wire List section contains net and pin connection information. Any information prior to the Component List section is ignored. Empty lines are also ignored. The Component List must precede the Wire List.

**Component List Section**

The Component List section contains part definitions, one per line. Each part definition has several fields:

- Part Number
- Tolerances
- Part Reference
- Part Name

The fields are all separated by white space. White space is one or more spaces between each field.

The Part Number field can contain any printable ASCII characters that are not a space, comma, or double quote as these characters are used in the AutoGen file formats.

The Tolerance field contains two part tolerances in percent, separated by one
or more spaces or commas. It’s best to use only one comma between the values. The values are in percent and can contain a following % sign after the number but it is not required.

The Part Reference field contains a part reference designator. This field must be unique to every part defined in the Component List section. The Part Reference field contains one or more alphabetic characters followed by one or more numeric characters. The import.dat file (described below) in the DIGALOG/include/autogen sub-directory contains information on parsing the part reference.

The Part Name is a field that is parsed. It begins with a numeric string (containing characters 0 through 9 with possibly a decimal point somewhere in between). The characters after can be used as multipliers if they are defined in the import.dat file. If the value is still zero, the number is replaced by the Default value from the import.dat file. The name doesn’t have to be a number and its value will be set to zero or the Default value from the import.dat file. It could also be an industry part number in the case of transistors, diodes and ICs. If the part is an industry part number and not a diode (as specified in the import.dat file), the part is defined as a component type of other.

The Component List section must be followed by the Wire List section.

Example Component List Section:

```plaintext
&lt;&lt; Component List &gt;&gt;
16089728 5,5  C32  .022u
16089728 5,5  C33  .022u
16089728 5,5  C34  .022u
16089728 5,5  C35  .022u
16089794 5,5  C38  1000p
16031261 5,5  R13  2.4K
16031261 5,5  R14  2.4K
16031261 5,5  R15  2.4K
16031261 5,5  R16  2.4K
16250144 1,1  Q1    NFET
16250144 1,1  Q2    NFET
```
Wire List Section

The Wire List section begins after the Component List section. Any lines that are empty are ignored. After the header line, a title line which must contain the text “NODE” (in upper case). After the headline and title line, two types of lines are parsed:

- Net Line
- Pin Line

For every Net Line, there is one or more Pin Lines associated. All Pin Lines following a Net Line belong to that Net Line.

A Net Line consists of two fields:

- Node Number
- Net Name

The Node Number is formatted as a number between brackets: [00001]. In reality it is ignored but must exist because it is used to determine that the current line is a net line. The fields in the net line are separated by one or more spaces. The Net Name is a string defining the name of the net. It can contain any characters but spaces, commas or double quotes. The length of the net name is unlimited but in practice it should be less than 64 characters.

The Pin Line consists of three fields:

- Part Reference
- Pin Number
- Pin Name

The Pin Line must contain spaces and/or tabs before the rest of the fields because that is how the Pin lines are differentiated from the Net Lines. All the fields are white space separated (one or more spaces or tabs).

The Part Reference field contains a part reference designator and one or more alphabetic characters followed by one or more numeric characters. It should match one of the Part References defined in the Component List section.
The Pin Number field contains a unique pin number on a part. It can be any combination of letters and/or numbers but typically they should be numbers since they are converted to numbers anyway.

The Pin Name field contains the name of the pin. Typically it is the same as the Pin Number field for two pin parts like resistors, capacitors, or inductors. The field can be any combination of letters and numbers.

All Pin Line fields after the last Net Line belong to that net.

Example Wire List Section:

```text
&lt;&lt; Wire List &gt;&gt;

NODE  REFERENCE   PIN #   PIN NAME

[00001] /BO OT VPP
   C48       1       1
   J1        B4      B4
   R22       1       1

[00002] /BO OT_VPP
   C43       1       1
   R21       1       1
   R22       2       2
   U1        91      91
   VIA00001  1       1
   VIA00002  1       1

[00003] /BRAKE_SW
   C8        1       1
   R26       2       2
   R31       1       1
   U1        67      67
   VIA00003  1       1
   VIA00004  1       1
```
Import Data File

The Import Data file is used to determine component types and value multipliers. This file is located in the DIGALOG/include/autogen sub-directory and is called import.dat. The import.dat file contains entries defining a match between component type and part reference leader. Other entries are the default value and zero or more follower multipliers. The default value and multipliers are used for the last component type entry parsed. Here is a sample import.dat file:

Resistance = R
  Default = 1
  K = 1000
  M = 1e6
 Capacitance = C,DC
  Default = 1e-6
  MFD = 1e-6
  MFDC = 1e-6
  U = 1e-6
  P = 1e-12
  UF = 1e-6
  PF = 1e-12
 DiodeV = CR,D
  Default = .7

Each line contains a name, an ‘=’ sign, and a value. The line either defines a new component type entry or a part name follower multiplier. The possible component types are:

Resistance
Capacitance
Inductance
DiodeV
Zener

The value for the component type is a comma-separated list of reference leaders. These reference leaders are one or more alphabetic characters (A through Z, no lower case).
If a name doesn't match a component type then it is either a default value or a multiplier. If the name is Default then the value of the entry is the default value for any part that has a value of zero. The rest of the entries after the default are the multipliers. A multiplier is a string after the value defined in the Part Name field in a wire list Component List section. A zero or more alphabetic characters that match a multiplier could follow a number in the Part Name field. In this case, the component value is multiplied by that multiplier. For example 1.0K defined for a reference leader R1 with the import.dat file above would import a resistance component type of 1000 ohms.

**Probe File**

The Probe File is used to assign channels to a one panel project. The file is generated by UniSoft Corporation’s PRONTO FIXTURE™ fixture design package. It looks for Probe Lines which contain 10 fields (separated by white space):

Part Reference
Pin Name
X
X location
Y
Y location
Channel
- (dash)
Net Name
Board Side

The only fields parsed are:

- Part Reference
- Pin Number
- Channel
- Net Name

Here is an example line:
The Part Reference should follow the rules in the Component List section and Wire List section. The Pin Name should follow the convention in the Wire List section. The Channel should be an integer numbers (a combination of 1 or more numeric characters). The Net Name should follow the convention in the Wire List section.

**Example Probe File:**

Prepared by UniSoft Corporation’s PRONTO FIXTURE(tm) fixture design package. For information, contact UniSoft at (203) 925-2980. Address: One Corporate Dr., Shelton, CT 06484

Probe Spacing: 100 mils

<table>
<thead>
<tr>
<th>Part</th>
<th>Channel</th>
<th>Pin Name</th>
<th>Channel</th>
<th>Channel</th>
<th>Net Name</th>
<th>Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>RV2</td>
<td>2</td>
<td>X</td>
<td>1.084</td>
<td>Y</td>
<td>1.362</td>
<td>1 - GROUND</td>
</tr>
<tr>
<td>RV1</td>
<td>1</td>
<td>X</td>
<td>0.743</td>
<td>Y</td>
<td>-1.351</td>
<td>35 - /IGN3</td>
</tr>
<tr>
<td>K1</td>
<td>2</td>
<td>X</td>
<td>0.254</td>
<td>Y</td>
<td>1.063</td>
<td>10 - /ENRELAY</td>
</tr>
<tr>
<td>K1</td>
<td>5</td>
<td>X</td>
<td>-0.219</td>
<td>Y</td>
<td>1.063</td>
<td>37 - U3_19</td>
</tr>
<tr>
<td>T7</td>
<td>1</td>
<td>X</td>
<td>-1.614</td>
<td>Y</td>
<td>-0.209</td>
<td>39 - /LFHOLD</td>
</tr>
<tr>
<td>T3</td>
<td>1</td>
<td>X</td>
<td>-0.747</td>
<td>Y</td>
<td>-0.209</td>
<td>42 - /LFREL</td>
</tr>
<tr>
<td>T5</td>
<td>1</td>
<td>X</td>
<td>-1.370</td>
<td>Y</td>
<td>1.500</td>
<td>49 - /LRHOLD</td>
</tr>
</tbody>
</table>
AutoGen CAE Wirelist Format
Appendix B - CAE Wirelist Format Definition For AutoGen

There are four files used with AutoGen at the present time. They are the Part File, Pin File, Map File and the Test File.

Engineering Notation

Some of the fields contained in various AutoGen files are formatted as engineering notation. This notation contains digits 0 through 9 and a single decimal point with a possible trailing multiplier character. There are several multipliers and are case insensitive except (M for mega and m for milli).

1. ‘T’ - tera
2. ‘G’ - giga
3. ‘M’ - mega
4. ‘k’ - kilo
5. ‘m’ - milli
6. ‘u’ - micro
7. ‘n’ - nano
8. ‘p’ - pico
9. ‘f’ - femto

Even though most of the multipliers are read case insensitively, they are formatted as single characters listed above during output with the same case. Here is an example:

1.200k

which is 1200.

NOTE 1: In the future these files may become sections in a single large file.

NOTE 2: All quoted strings can have 200+ characters. Use good judgment when specifying quoted string. A good size is less than 32 characters.

Part File Format

The Part File (.prt) is the file that AutoGen stores all of the part information. The file contains a header line and one or more data lines. The information stored in this file is used to cross reference what is stored in the Pin File (.pin), and the Test File (.tst).
Header Line

The Header Line is always the first line in the file. It looks like:

```
# AutoGen Part File - 2
```

The number after the - is the version of the file format. For this document version 2 is defined. The first part (# AutoGen Part File -) of the string MUST exist for AutoGen to recognize it as a Part File.

Data Line

All of the lines after the Header Line are Data Lines. Each Data Line contains a title and data. The title just defines what the line is and the data is specific to the title. There is a ‘:' separator between the title and the data. The ‘:' must be right next (no spaces) to the title (i.e. PART:) for the title to be recognized. Here is a sample Part Data Line:

```
PART: “R1_0062A”, “10”, “0025”, “Resistance”, 10.00, 20.00, 5.000, “”
```

This is a part with reference “R1_0062A” with part name “10” and part number “0025”. It is measured by a “Resistance” type measurement with a value of 10 and +20% and -5% tolerances without a comment (“”).

Part Data

A Part Data line is defined by the title ‘PART:’. The Part Data portion of the Data Line contains 8 fields:

1. Part Reference
2. Part Name
3. Part Number
4. Component Type
5. Part Value
6. Part Plus Tolerance
7. Part Minus Tolerance
8. Comment

Each field is separated by a single ‘,’ with possible white space around it. White space is defined as one or more spaces (‘ ’). There are 5 quoted fields: Part Reference, Part Name, Part Number, Component Type, and Comment. Quoted fields allow any printable ASCII characters in them except a quote (“). The other 3 fields: Part Value, Part Plus Tolerance and Part Minus Tolerance allow engineering formatted numbers.
Part Reference

The Part Reference field is a quoted string. This field is used to cross reference this part with the net list and the test list. It can contain any printable ASCII characters except ‘”’.

Part Name

The Part Name field is also a quoted string. This field is used usually during importing to determine the Part Value field. After importing, this field is not needed by AutoGen but is usually used for display purposes. This field can contain any printable ASCII characters except ‘”’.

Part Number

The Part Number field is a quoted string. It is usually the company part number and is not used by AutoGen. It can be used for output during in-circuit tests though. It can contain any printable ASCII characters except ‘”’.

Component Type

The Component Type field is used to determine the test type of the part. It can be any of the following values:

1. Resistance
2. Capacitance
3. Inductance
4. DiodeV
5. DiodeI
6. Zener
7. PotAdjust
8. Switch
9. Other

These values are case insensitive. Any other values are set automatically to Other when read by AutoGen.

Part Value

The part value is formatted in engineering notation (as described above). It is the actual value of the component.

Part Plus Tolerance

The Part Plus Tolerance field is also formatted in engineering notation (as described above). It is specified in percent (i.e. 10 = 10%).
**Part Minus Tolerance**

The Part Plus Tolerance field is also formatted in engineering notation (as described above). It is specified in percent (i.e. \(10 = 10\%\)). This should be a positive value with out a sign character.

**Comment**

The Comment field is also a quoted string. This field can contain any printable ASCII characters except ‘“’. It is unused by AutoGen except for the user.

Here is an example AutoGen .prt file:

```
# AutoGen Part File - 2
PART: "P8", "0062A BOARD", "**************", "Other", 0.000, 0.000, 0.000, ""
PART: "C4_0062A", "12PF", "5007", "Capacitance", 45.000p, 50.00, 20.00, ""
PART: "C3_0062A", "100PF", "5010", "Capacitance", 100.0p, 80.00, 20.00, ""
PART: "C2_0062A", "0.01UF", "5040", "Capacitance", 10.00n, 80.00, 20.00, ""
PART: "C1_0062A", "1.0UF", "5199", "Capacitance", 1.010u, 80.00, 20.00, ""
PART: "R_PS1_0062A", "15", "9000", "Resistance", 15.00, 30.00, 30.00, ""
PART: "R1_0062A", "10", "0025", "Resistance", 10.00, 20.00, 5.000, ""
PART: "D1_0062A", "HSMS-2800", "0813", "DiodeV", 250.0m, 20.00, 20.00, ""
PART: "Q1_0062A", "MOSFET", "1301", "Resistance", 5.000, 25.00, 25.00, ""
```

**Pin File Format (Netlist Format)**

The Pin File (.pin) is the file that AutoGen stores all of the net list information. The file contains a header line and one or more data lines. The information stored in this file is used to cross reference what is stored in the Part File (.prt), and the Test File (.tst).

**Header Line**

The Header Line is always the first line in the file. It looks like:

```
# AutoGen Pin File - 2
```

The number after the - is the version of the file format. For this document version 2 is defined. The first part (# AutoGen Pin File -) of the string MUST exist for AutoGen to recognize it as a Pin File.

**Data Line**

All of the lines after the Header Line are Data Lines. Each Data Line contains a title and data. The title just defines what the line is and the data is specific to
the title. There is a ‘:’ separator between the title and the data. The ‘:’ must be right next to the title (i.e. NET:) for the title to be recognized. Here are sample lines for one net:

NET: “ICPOSIN_1_0062A”, 2, 3, 10.00 , 1.000k, “”
PIN: “Q1_0062A”, “G”, 1
PIN: “R1_0062A”, “2”, 2

This is a net with name “ICPOSIN_1_0062A” with 2 pins and a shorts test type of both shorts and opens. The first pin has part reference “Q1_0062A” with pin name of “G” and pin number 1. The second pin has part reference “R1_0062A” with a pin name of “2” and a pin number of 2.

There are two types of data lines: Net data and Part Pin data. Each Net data line has zero or more Pin data lines. These lines are the part pins tied to the net.

**Net Data line**

The Net Data line is defined by the title ‘NET:’. It contains 7 fields:

1. Net Name
2. Number Of Pins
3. Shorts Test Type
4. Shorts Threshold
5. Opens Threshold
6. Test Time
7. Comment

Each field is separated by a ‘,’ with possible white space (‘ ‘). The Net Name field is a quoted string. The Number Of Pins and the Shorts Test Type are integers. The Net Name field can contain any printable ASCII characters except “”. The Number Of Pins can contain any ASCII digits 0 through 9. The Shorts Test Type has 4 possible values (0 through 3):

- 0 = No Shorts Test
- 1 = Shorts Only Test
- 2 = Opens Only Test
- 3 = Shorts And Opens Test

The Shorts Threshold, Opens Threshold, and Test Time fields are in engineering format described above. The Comment field is a quoted string.
**Part Pin Data line**

The Part Pin line is defined by the title ‘PIN:’. It contains 3 fields:

1. Part Reference
2. Pin Name
3. Pin Number

Each field is separated by a ‘,’ with possible white space (' ') around it. The Part Reference field and the Pin Name field are a quoted strings. The Pin Number field is an integer. The Part Reference field and the Pin Name field can contain any printable ASCII characters except ‘‘‘. The Pin Number field can contain any ASCII digits 0 through 9. The '.pin' file should not have duplicate pins for a single part except if their number is 0 which will get automatically assigned to the next available pin. Here is an example AutoGen .pin file:

```
# AutoGen Net File - 2
NET: "ICPOSIN_1_0062A", 2, 3, 10.00 , 1.000k, ""
PIN: "Q1_0062A", "G", 1
PIN: "R1_0062A", "2", 2
NET: "POSRAILS_1_0062A", 4, 2, 10.00 , 1.000k, ""
PIN: "Q1_0062A", "D", 2
PIN: "C1_0062A", "1", 1
PIN: "C2_0062A", "1", 1
PIN: "D1_0062A", "1", 1
NET: "CKTOUPTPUT1_0062A", 2, 3, 10.00 , 1.000k, ""
PIN: "R_PS1_0062A", "1", 1
PIN: "C4_0062A", "1", 1
NET: "AMPINPUT1_0062A", 1, 3, 10.00 , 1.000k, ""
PIN: "C3_0062A", "1", 1
NET: "AMPIONMP1_0062A", 1, 3, 10.00 , 1.000k, ""
PIN: "C4_0062A", "2", 2
NET: "N000060062A", 2, 2, 10.00 , 1.000k, ""
PIN: "Q1_0062A", "S", 3<
PIN: "R1_0062A", "1", 1
NET: "AMPOUTPUT1_0062A", 2, 3, 10.00 , 1.000k, ""
PIN: "R_PS1_0062A", "2", 2
PIN: "D1_0062A", "A", 2
NET: "SGND_0062A", 3, 3, 10.00 , 1.000k, ""
PIN: "C2_0062A", "2", 2
PIN: "C1_0062A", "2", 2
PIN: "C3_0062A", "2", 2
```

— — — — Cut Here — — — —
Map File Format

The Map File (.map) is the file that AutoGen stores all of the net to channel mapping information. This is done to allow panelizing of boards and this file contains the channel mapping for each board of a panel. The file contains a header line and one or more data lines.

Header Line

The Header Line is always the first line in the file. It looks like:

```
# AutoGen Map File - 1
```

Data Line

All of the lines after the Header Line are Data Lines. Each Data Line contains a title and data. The title just defines what the line is and the data is specific to the title. There is a ‘:’ separator between the title and the data. The ‘:’ must be right next (no spaces) to the title (i.e. NETNAME:) for the title to be recognized. Here a sample line for one map:

```
AMPINPUT_1_0062A: 3,-1
```

Here the net “AMPINPUT_1_0062A” contains channel 3 and Kelvin channel (-1).

The title portion of a data line is the net name. The data portion contains integer pairs of numbers: one for each board in a panel. Each integer is separated by a ‘,’ with possible white space (‘ ’) around it. If any number in the pair is a -1, it is an unassigned channel. Each pair of integers is a measure channel and a kelvin channel for a board on the net.
**Test File Format**

The Test File (.tst) is the file that AutoGen stores all of the test information. The file contains a header line and one or more data lines.

**Header Line**

The Header Line is always the first line in the file. It looks like:

```
# AutoGen Test File - 2
```

**Data Line**

All of the lines after the Header Line are Data Lines. Each Data Line contains a title and data. The title just defines what the line is and the data is specific to the title. There is a ‘:’ separator between the title and the data. The ‘:’ must be right next (no spaces) to the title (i.e. TEST:) for the title to be recognized. There are two types of data lines in a test file: a test data line and a guard data line. There is either zero or one guard data line for each test line.

**Test Data Line**

A Test Data line is defined by the title ‘TEST:’. The Test Data line is a line containing test information for a specific part in the AutoGen part file. Their 8 fields in a test data line:

1. Part Reference
2. Forcing Value
3. Trip Point
4. Test Time
5. Source Pin
6. Measure Pin
7. Number Of Guards
8. Test Mode

The Part Reference field is a quoted string which can contain any valid printable ASCII character except a quote ("). The Forcing Value, Trip Point, and Test Time are in engineering format described above. The Source Pin, Measure Pin, Number Of Guards, and Test Mode are all positive integer values. Each field is separated by a ‘,’ with possible white space surrounding it.

**Part Reference**

The Part Reference field is a part reference that must exist in the AutoGen Part File (.prt). This is the link between the test data and the part.
Forcing Value, Trip Point, and Test Time

The Forcing Value, Trip Point and Test Time fields are engineering notation formatted numbers (described above) that is used for executing a test for a specific component. The units and valid values for these fields are dependent upon the component type (in the AutoGen Part file). These are described in the functional call section of the manual.

Source Pin and Measure Pin

The source and measure pins are positive integer pin numbers of the part. These then get referenced to a net and then a channel number.

Number Of Guards

The number of guards field is a positive integer (including 0) that define how many guards are used in the current test. If this field is non-zero, then there is a Guard Data Line after the Test Data Line (described below).

Test Mode

The Test Mode field can be either 0 or 1. When the field is 0 the test is disabled. When the field is 1 the test is enabled. A test enabled will be allowed to be executed and included in the generated code. A test that is disabled is ignored during execution and generation.

Guard Data Line

The Guard Data Line is defined with a title of ‘GUARDS:’. It contains a ‘,’ (with possible white space surrounding) separated list of net names. The net names are all quoted strings and may contain any visible ASCII character except (“). An example test file is shown on the next page.
# AutoGen Test File - 2

TEST: "C4_0062A", 1.500k, 300.0m, 1.000m, 1, 2, 1, 1
GUARDS: AMPOUTPUT_1_0062A,

TEST: "C3_0062A", 1.500k, 300.0m, 1.000m, 2, 1, 1, 1
GUARDS: POSRAIL_1_0062A,

TEST: "C2_0062A", 500.0 , 300.0m, 1.000m, 1, 2, 0, 0
TEST: "C1_0062A", 500.0 , 300.0m, 1.000m, 2, 1, 0, 1
GUARDS: AMPCOMP_1_0062A,

TEST: "R_PS1_0062A", 1.000 , 300.0m, 1.000m, 2, 1, 1, 1
GUARDS: AMPCOMP_1_0062A,

TEST: "R1_0062A", 1.000 , 300.0m, 1.000m, 1, 2, 0, 1
TEST: "D1_0062A", 500.0u, 3.000 , 1.000m, 2, 1, 0, 1
TEST: "Q1_0062A", 500.0m, 300.0m, 1.000m, 3, 2, 0, 1
TEST: "C4_0062B", 1.500k, 300.0m, 1.000m, 1, 2, 1, 1

——— Cut Here ————
Sample.frm
Appendix C - CYX32.frm

VERSION 5.00
Object = "{0BA686C6-F7D3-101A-993E-0000C0EF6F5E}\#1.0\#0";
"THREED32.OCX"
Object = "{3B7C8863-D78F-101B-B9B5-04021C009402}\#1.1\#0";
"RICHTX32.OCX"
Begin VB.Form Executive
Appearance = 0 'Flat
BackColor = &H00C0C0C0&
Caption = "Test Executive"
ClientHeight = 5355
ClientLeft = 2145
ClientTop = 2295
ClientWidth = 7770
BeginProperty Font
Name = "MS Sans Serif"
Size = 8.25
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
ForeColor = &H80000008&
LinkTopic = "Form1"
PaletteMode = 1 'UseZOrder
ScaleHeight = 5355
ScaleWidth = 7770
Begin Threed.SSPanel pnlCYX
Align = 1 'Align Top
Height = 1020
Index = 0
Left = 0
TabIndex = 5
Top = 0
Width = 7770
_Version = 65536
_ExtentX = 13705
_ExtentY = 1799
_StockProps = 15
ForeColor = 8388608
BackColor = -2147483633

BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
  Name = "MS Sans Serif"
  Size = 8.25
  CharSet = 0
  Weight = 700
  Underline = 0 'False
  Italic = 0 'False
  Strikethrough = 0 'False
EndProperty

BevelOuter = 0

Begin VB.CommandButton cmdStop
  Caption = "&Stop"
  Enabled = 0 'False
  BeginProperty Font
    Name = "MS Sans Serif"
    Size = 24
    CharSet = 0
    Weight = 700
    Underline = 0 'False
    Italic = 0 'False
    Strikethrough = 0 'False
  EndProperty
  Height = 690
  Left = 4020
  TabIndex = 1
  Top = 120
  Width = 3450
End

Begin VB.CommandButton cmdStart
  Caption = "&Start"
  BeginProperty Font
    Name = "MS Sans Serif"
    Size = 24
    CharSet = 0
    Weight = 700
    Underline = 0 'False
    Italic = 0 'False
    Strikethrough = 0 'False
  EndProperty
  Height = 690
  Left = 4020
  TabIndex = 1
  Top = 120
  Width = 3450
End
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
Height = 690
Left = 195
TabIndex = 0
Top = 120
Width = 3420
End

Begin Threed.SSPanel pnlCYX
Align = 1 'Align Top
Height = 10125
Index = 2
Left = 0
TabIndex = 3
Top = 1650
Width = 7770
_Version = 65536
_ExtentX = 13705
_ExtentY = 17859
_StockProps = 15
ForeColor = 8388608
BackColor = -2147483633
BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
Name = "MS Sans Serif"
Size = 8.25
Charset = 0
Weight = 400
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
BevelOuter = 0
Begin RichTextLib.RichTextBox txtScreen
Height = 3075
Left = 180
End
TabIndex        =   6
Top             =   420
Width           =   7395
_ExtentX        =   13044
_ExtentY        =   5424
_Version        =   327680
Enabled         =   -1  'True
ScrollBars      =   3

BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
  Name            =   “Courier New”
  Size            =   8.25
  Charset         =   0
  Weight          =   400
  Underline       =   0  ‘False
  Italic          =   0  ‘False
  Strikethrough   =   0  ‘False
EndProperty

End

Begin RichTextLib.RichTextBox txtPrinter
  Height          =   675
  Left            =   180
  TabIndex        =   7
  Top             =   3720
  Visible         =   0  ‘False
  Width           =   795
  _ExtentX        =   1402
  _ExtentY        =   1191
  _Version        =   327680
  ScrollBars      =   3

BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
  Name            =   “Tahoma”
  Size            =   8.25
  Charset         =   0
  Weight          =   400
  Underline       =   0  ‘False
  Italic          =   0  ‘False
  Strikethrough   =   0  ‘False
EndProperty

End
Begin RichTextLib.RichTextBox txtLogFile
   Height = 675
   Left  = 1200
   TabIndex = 8
   Top    = 3720
   Visible = 0 'False
   Width  = 795
   _ExtentX = 1402
   _ExtentY = 1191
   _Version = 327680
   Enabled = -1 'True
   ScrollBars = 3
BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
   Name = "Tahoma"
   Size  = 8.25
   Charset = 0
   Weight = 400
   Underline = 0 'False
   Italic = 0 'False
   Strikethrough = 0 'False
EndProperty
End
Begin VB.Label lblResults
   Appearance = 0 'Flat
   AutoSize = -1 'True
   BackColor = &H80000005&
   BackStyle = 0 'Transparent
   Caption = "Results:"
   ForeColor = &H00FFFFFF&
   Height = 195
   Left = 180
   TabIndex = 4
   Top = 120
   Width = 720
End
End
Begin Threed.SSPanel pnlCYX
   Align = 1 'Align Top
   Height = 630
Index       =   1
Left         =   0
TabIndex     =   2
Top          =   1020
Width        =   7770
_Version     =   65536
_ExtentX     =   13705
_ExtentY     =   1111
_StockProps  =   15
ForeColor    =   16777215
BackColor    =   -2147483633
BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
    Name                  =   “MS Sans Serif”
    Size                  =   18
    Charset                =   0
    Weight                 =   700
    Underline            =   0  ‘False
    Italic                =   0  ‘False
    Strikethrough         =   0  ‘False
EndProperty
BevelOuter   =   0
End
Begin VB.Menu mnuFile
    Caption                  =   “&File”
Begin VB.Menu mnuConfigure
    Caption                  =   “&Output”
Begin VB.Menu mnuScreen
    Caption                  =   “&Screen”
End VB.Menu mnuScreenOutput
    Caption                  =   “&No Results”
    Index                    =   0
End VB.Menu mnuScreenOutput
Begin VB.Menu mnuScreenOutput
    Caption                  =   “&Fail Data”
    Checked                  =   -1  ‘True
    Index                    =   1
End
Begin VB.Menu mnuScreenOutput
    Caption         =   "&All Data"
    Index           =   2
End
End
Begin VB.Menu mnuPrinter
    Caption         =   "&Printer"
    Begin VB.Menu mnuPrinterOutput
        Caption         =   "&No Results"
        Checked         =   -1 'True
        Index           =   0
    End
    Begin VB.Menu mnuPrinterOutput
        Caption         =   "&Fail Data"
        Index           =   1
    End
    Begin VB.Menu mnuPrinterOutput
        Caption         =   "&All Data"
        Index           =   2
    End
End
End
Begin VB.Menu mnuLogFile
    Caption         =   "&Log File"
    Begin VB.Menu mnuLogFileOutput
        Caption         =   "&No Results"
        Checked         =   -1 'True
        Index           =   0
    End
    Begin VB.Menu mnuLogFileOutput
        Caption         =   "&Fail Data"
        Index           =   1
    End
    Begin VB.Menu mnuLogFileOutput
        Caption         =   "&All Data"
        Index           =   2
    End
End
End
Begin VB.Menu mnuLogFile
    Caption         =   "&Log File"
    Begin VB.Menu mnuLogFileOutput
        Caption         =   "&No Results"
        Checked         =   -1 'True
        Index           =   0
    End
    Begin VB.Menu mnuLogFileOutput
        Caption         =   "&Fail Data"
        Index           =   1
    End
    Begin VB.Menu mnuLogFileOutput
        Caption         =   "&All Data"
        Index           =   2
    End
End
End
Begin VB.Menu mnuFileSep
    Caption         =   “-”
End
Begin VB.Menu mnuFileExit
    Caption         =   “E&xit”
End
End
End
Attribute VB_Name = “Executive”
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
‘ File: executive.frm
‘ $Revision: 1.4 $
‘ Option Explicit


Private Const CYXINI = “cyx.ini”
Private Const SECTION = “Output”
Private Const SCREENKEY = “Screen”
Private Const PRINTERKEY = “Printer”
Private Const LOGFILEKEY = “LogFile”

Private Const NORESULTS = 0
Private Const FAILDATA = NORESULTS + 1
Private Const ALLDATA = FAILDATA + 1
Private Const LASTRESULTS = ALLDATA

Private EXECINIFILE As String
Public ScreenTB As New RichTextBoxEx
Public PrinterTB As New RichTextBoxEx
Public LogFileTB As New RichTextBoxEx
Public LogFileMask As Integer
Public PrinterMask As Integer
Public ScreenMask As Integer
Public Interrupted As Integer

Private Const MAXSTRLEN = 50

Private Sub CVSInfo()
    Dim s As String
    s = Header
End Sub

Private Sub cmdStart_Click()
    Dim LastResult As Integer
    Dim cnt As Integer

    'If cmdStart.Caption = “Continue” Then
    '    NextTest = True
    '    Exit Sub
    'End If

    On Error GoTo ExecuteHandler

    txtScreen.Text = ""
    txtPrinter.Text = ""
    txtLogFile.Text = ""

    'disable/enable buttons (botones de inhabilitación/habilitación)
    cmdStart.Enabled = False
    cmdStop.Enabled = True
    cmdStop.SetFocus

    pnlCYX(1) = "Status: Testing"
    pnlCYX(0).BackColor = &H8000000F
    pnlCYX(1).BackColor = &H8000000F
    pnlCYX(2).BackColor = &H8000000F
    Interrupted = False
LastResult = ExecuteCYX()

ExecuteHandler:
   If Err.Number Then
      MsgBox "Error: " & CStr(Err.Number) & " ", vbOK
   End If
   On Error Resume Next

   'Check for failures ("Verificar los fallos")
   If LastResult = IFAIL Or Interrupted Then
      pnlCYX(1) = "Status: FAIL"
      pnlCYX(0).BackColor = &H80&
      pnlCYX(1).BackColor = &H80&
      pnlCYX(2).BackColor = &H80&
   Else
      pnlCYX(1) = "Status: PASS"
      pnlCYX(0).BackColor = &H8000&
      pnlCYX(1).BackColor = &H8000&
      pnlCYX(2).BackColor = &H8000&
   End If
   Me.Refresh

   'disable/enable buttons (botones de inhabilitación/habilitación)
   cmdStop.Enabled = False
   cmdStart.Enabled = True
   cmdStart.Caption = "&Start"
   cmdStart.SetFocus

   If PrinterMask = ALLDATA Or (LastResult = IFAIL And PrinterMask = FAILDATA) Then
      FlushPrinter
   End If

   If LogFileMask = ALLDATA Or (LastResult = IFAIL And LogFileMask = FAILDATA) Then
      FlushLogFile
   End If

End Sub
Private Sub cmdStop_Click()
    Interrupted = True
End Sub

Private Sub Form_Load()

    ‘set up printer logging parameter (ajustar el parámetro de registro de la impresora)
    PrinterMask = GetSetting(CYXINI, SECTION, PRINTERKEY, NORESULTS)
    mnuPrinterOutput_Click PrinterMask

    ‘set up file logging parameter (ajustar el parámetro de registro del archivo)
    LogFileMask = GetSetting(CYXINI, SECTION, LOGFILEKEY, NORESULTS)
    mnuLogFileOutput_Click LogFileMask

    ‘set up screen logging parameter (ajustar el parámetro de registro de la pantalla)
    ScreenMask = GetSetting(CYXINI, SECTION, SCREENKEY, FAILDATA)
    mnuScreenOutput_Click ScreenMask

    txtScreen.Text = ""
    Set ScreenTB.TB = txtScreen
    txtPrinter.Text = ""
    Set PrinterTB.TB = txtPrinter
    txtLogFile.Text = ""
    Set LogFileTB.TB = txtLogFile

    InitializeCYX
End Sub

Private Sub Form_Resize()
    Dim w As Single
    Dim h As Single
    Dim dw As Single
    Dim dh As Single
    w = Executive.ScaleWidth
    h = Executive.ScaleHeight - pnlCYX(2).Top
    pnlCYX(2).Move pnlCYX(2).Left, pnlCYX(2).Top, w, h
    dw = txtScreen.Left - pnlCYX(2).Left
dh = txtScreen.Top - pnlCYX(2).Top
txtScreen.Move txtScreen.Left, txtScreen.Top, _
    w - 2 * txtScreen.Left, h - 2 * txtScreen.Top
End Sub

Private Sub Form_Unload(Cancel As Integer)
    End
End Sub

Private Sub mnuFileExit_Click()
    Unload Executive
End Sub

Private Sub mnuLogFileOutput_Click(Index As Integer)
    Dim i As Integer
    For i = NORESULTS To LASTRESULTS
        mnuLogFileOutput(i).Checked = vbUnchecked
    Next i
    mnuLogFileOutput(Index).Checked = vbChecked
    SaveSetting CYXINI, SECTION, LOGFILEKEY, Index
    LogFileMask = Index
End Sub

Private Sub mnuPrinterOutput_Click(Index As Integer)
    Dim i As Integer
    For i = NORESULTS To LASTRESULTS
        mnuPrinterOutput(i).Checked = vbUnchecked
    Next i
    mnuPrinterOutput(Index).Checked = vbChecked
    SaveSetting CYXINI, SECTION, PRINTERKEY, Index
    PrinterMask = Index
End Sub

Private Sub mnuScreenOutput_Click(Index As Integer)
    Dim i As Integer
    For i = NORESULTS To LASTRESULTS
        mnuScreenOutput(i).Checked = vbUnchecked
    Next i
    mnuScreenOutput(Index).Checked = vbChecked
SaveSetting CYXINI, SECTION, SCREENKEY, Index
ScreenMask = Index
End Sub

Public Sub LogHeader(s As String)
Dim l As Long
l = ScreenTB.LineCount()
ScreenTB.Append l, vbCrLf & s
l = PrinterTB.LineCount()
PrinterTB.Append l, vbCrLf & s
l = LogFileTB.LineCount()
LogFileTB.Append l, vbCrLf & s
End Sub

Public Sub Log(s As String, cond As Integer)
Dim l As Long
Dim sel As Long
If ScreenMask = ALLDATA Or (cond = IFAIL And ScreenMask = FAILDATA) Then
l = ScreenTB.LineCount()
ScreenTB.Append l, vbCrLf & IIf(cond = IFAIL, "FAIL", "    ") & " " & s
End If
If PrinterMask = ALLDATA Or (cond = IFAIL And PrinterMask = FAILDATA) Then
l = PrinterTB.LineCount()
PrinterTB.Append l, vbCrLf & IIf(cond = IFAIL, "FAIL", "    ") & " " & s
End If
If LogFileMask = ALLDATA Or (cond = IFAIL And LogFileMask = FAILDATA) Then
l = LogFileTB.LineCount()
LogFileTB.Append l, vbCrLf & IIf(cond = IFAIL, "FAIL", "    ") & " " & s
End If
End Sub

Public Sub FlushPrinter()
Dim fn As Integer

On Error GoTo FlushResultsHandler
If txtPrinter <> "" Then
    txtPrinter.SelStart = 0
    txtPrinter.SelLength = Len(txtPrinter.TextRTF)
    Printer.Print ""
    txtPrinter.SelPrint Printer.hDC
    Printer.EndDoc
End If

FlushResultsHandler:
    If Err Then
        MsgBox "Printer Error: " & Err, vbOK
        ' err.description
    End If
    Exit Sub
End Sub

Public Sub FlushLogFile()
    Dim fn As Integer
    Dim fname As String

    On Error GoTo FlushLogFileHandler

    If txtLogFile <> "" Then
        fname = Format(Date, "mmmdd-yy") & ".log"
        fn = FreeFile
        Open fname For Append As fn
        Print #fn, txtLogFile.Text
        Close fn
    End If

FlushLogFileHandler:
    If Err Then
        MsgBox "LogFile Error: " & Err, vbOK
    End If
    Exit Sub
End Sub
agresult.bas
Appendix D - agresult.bas

Attribute VB_Name = “AutoGenResults”
‘ File: agresult.bas
‘ $Revision: 1.25 $

Option Explicit

Public Well As Integer  ‘ this defaults to 0
Public shorts As Collection
Public disabled As Collection
Public maxshorts As Integer

Private Sub CVSInfo()
    Dim s As String
    s = Header
End Sub

Public Sub Main()
    Executive.Show
End Sub

Public Sub BeforeCyxTests(cyxd As CyxData)

    IcamClear
    InitializeCalibration cyxd
    Fixture IIf(Well = 0, DUTL, DUTR), DUTLower
    maxshorts = 10

End Sub

Public Sub AfterCyxTests(cyxd As CyxData)

    Fixture IIf(Well = 0, DUTL, DUTR), DUTRaise
    ‘ Well = 1 - Well
End Sub
Public Sub BeforeAutogenTests(agd As AutoGenData)
    Dim FixtureId As Integer
    Dim f1 As Integer
    Dim f2 As Integer
    Dim Timeout As Integer

    FixID FixtureId
    If FixtureId <> agd.FixtureId Then
        Executive.LogHeader "Read FixtureID(" & Hex(FixtureId) & ") is not the project FixtureID(" & Hex(agd.FixtureId) & ")"
        agd.PassOp = ISTOP
        agd.FailOp = ISTOP
        agd.Condition = IFAIL
    End If

    Timeout = 10 ' 10 10ths of a second or 1 second
    Do While Timeout <> 0
        FixStat f1, f2
        If Well = 0 Then
            If f1 = 1 Then Exit Do
        Else
            If f2 = 1 Then Exit Do
        End If
        Idle 100
        Timeout = Timeout - 1
    Loop
    If Timeout = 0 Then
        Executive.LogHeader "Fixture status timeout, can’t run test"
        agd.PassOp = ISTOP
        agd.FailOp = ISTOP
        agd.Condition = IFAIL
    End If
End Sub

Public Sub AfterAutogenTests(agd As AutoGenData)
End Sub

Public Sub BeforeBoard(agd As AutoGenData)
    Executive.LogHeader "Board " & CStr(agd.BoardNumber)
Set shorts = New Collection
Set disabled = New Collection
End Sub

Public Sub AfterBoard(agd As AutoGenData)
If Executive.Interrupted = True Then
    agd.PassOp = ISTO P
    agd.FailOp = ISTO P
End If
If shorts.Count > 2 And shorts.Count < maxshorts Then
    Executive.LogHeader "Shorts Diagnostic"
    ShortsDiagnostic shorts, disabled
End If
End Sub

Public Sub BeforeAllShorts(agd As AutoGenData)
    Executive.LogHeader "Shorts Test"
End Sub

Public Sub AfterAllShorts(agd As AutoGenData)
End Sub

Public Sub BeforeAllOpens(agd As AutoGenData)
    Executive.LogHeader "Opens Test"
End Sub

Public Sub AfterAllOpens(agd As AutoGenData)
End Sub

Public Sub BeforeAllParts(agd As AutoGenData)
    Executive.LogHeader "Parts Test"
End Sub

Public Sub AfterAllParts(agd As AutoGenData)
End Sub
Public Sub BeforePartTest(pd As PartData)

End Sub

Public Sub AfterPartTest(pd As PartData)
Dim s As String

s = Format(pd.Reference, "!@@@@@@@@@@") & 
   " S=" & CStr(pd.SourceChannel.Item(pd(BoardNumber)) & 
   " M=" & CStr(pd.MeasureChannel.Item(pd(BoardNumber)) & 
   " Exp=" & DoubleToEngNotation(pd.ExpectedValue) & 
   " +=" & DoubleToEngNotation(pd.PlusExpected) & 
   " -=" & DoubleToEngNotation(pd.MinusExpected) & 
   " Act=" & DoubleToEngNotation(pd.ActualValue)

Executive.Log s, pd.Condition

DoEvents
If Executive.Interrupted = True Then
   pd.PassOp = ISTOP
   pd.FailOp = ISTOP
End If
End Sub

Public Sub BeforeUserTests(ags As AutoGenData)

End Sub

Public Sub AfterUserTests(ags As AutoGenData)

End Sub

Public Sub BeforeUserTest(pd As PartData)

End Sub

Public Sub AfterUserTest(pd As PartData)

End Sub
Public Sub BeforeShortTest(sd As ShortsData)

End Sub

Public Sub AfterShortTest(sd As ShortsData)
Dim s As String
    s = Format(sd.NetName, "!@@@@@@@@@@") & 
        " Chan=" & CStr(sd.Channel.Item(sd.BoardNumber)) & _
        " Res=" & DoubleToEngNotation(sd.Resistance) & _
        " < Exp=" & DoubleToEngNotation(sd.ShortsThreshold)

    Executive.Log s, sd.Condition

    If sd.Condition = IFAIL Then
        shorts.Add sd
    End If

    If sd.TestType <> nttShorts Or sd.TestType <> nttShortsOpens Then
        disabled.Add sd
    End If

    DoEvents
    If Executive.Interrupted = True Then
        sd.PassOp = ISTOP
        sd.FailOp = ISTOP
    End If
End Sub

Public Sub BeforeOpenPinTest(sd As ShortsData)

End Sub

Public Sub AfterOpenPinTest(sd As ShortsData)
Dim s As String
    s = Format(sd.NetName, "!@@@@@@@@@@") & 
        " Chan=" & CStr(sd.Channel.Item(sd.BoardNumber)) & _
        " Res=" & DoubleToEngNotation(sd.Resistance) & _
        " > Exp=" & DoubleToEngNotation(sd.OpensThreshold)
Executive.Log s, sd.Condition

DoEvents
If Executive.Interrupted = True Then
    sd.PassOp = ISTOP
    sd.FailOp = ISTOP
End If
End Sub

Public Sub ShortsDiagnostic(sc As Collection, dc As Collection)
    Dim i As Integer
    Dim j As Integer
    Dim sdi As ShortsData
    Dim sdj As ShortsData
    Dim chi As Integer
    Dim chj As Integer
    Dim minres As Double
    Dim maxtesttime As Double
    Dim measres As Double
    Dim s As String
    Executive.LogHeader “Shorts Diagnostic Test”

    ' Loop through each short test failure
    For i = 1 To sc.Count - 1
        Set sdi = sc.Item(i)
        chi = sdi.Channel.Item(sdi.BoardNumber)
        Guard chi

        ' check between other shorts failures
        For j = i + 1 To sc.Count
            Set sdj = sc.Item(j)
            chj = sdj.Channel.Item(sdj.BoardNumber)
            maxtesttime = IIf(sdi.TestTime > sdj.TestTime, _
                               sdi.TestTime, sdj.TestTime)
            minres = 1.2 * IIf(sdi.Resistance > sdj.Resistance, _
                               sdi.Resistance, sdj.Resistance)
            ' minres = 2% lowest measured resistance
        Next j
    Next i
End Sub
' take measurement
ShortsMeas measres, chj, 0.3, 0.3, maxtesttime

' if the measurement < minres Then two nets are shorted
If measres < minres Then
    s = "Nets: " & sdi.NetName & " and " & sdj.NetName & " are shorted"
    Executive.Log s, sdi.Condition
End If

Next j

' check between disabled shorts failures
For j = 1 To dc.Count
    Set sdj = dc.Item(j)
    chj = sdj.Channel.Item(sdj.BoardNumber)
    maxtesttime = IIf(sdi.TestTime > sdj.TestTime, _
                      sdi.TestTime, sdj.TestTime)
    ' minres = 2% lowest measured resistance
    minres = 1.2 * IIf(sdi.Resistance > sdj.Resistance, _
                      sdj.Resistance, sdj.Resistance)

    ' take measurement
    ShortsMeas measres, chj, 0.3, 0.3, maxtesttime

    ' if the measurement < minres Then two nets are shorted
    If measres < minres Then
        s = "Nets: " & sdi.NetName & " and " & sdj.NetName & " are shorted"
        Executive.Log s, sdi.Condition
    End If

Next j

Unguard chj
Next i

Executive.LogHeader "Shorts Diagnostic Test Done"
End Sub

Public Sub InitializeCalibration(cyxd As CyxData)
Dim i As Integer
Dim o As Object
Dim agr As AutoGenRoot
Dim pt As PartsTest
Dim pd As PartData
Dim c As Double
Dim s As String

' search For AutoGenData object
For Each o In cyxd.Test
  If TypeName(o) = "AutoGenRoot" Then
    Set agr = o
    Exit For
  End If
Next

If agr Is Nothing Then
  Executive.LogHeader "No AutoGenRoot Found"
  Exit Sub
End If

' search For PartTest object
For Each o In agr.cyxd.Test
  If TypeName(o) = "PartsTest" Then
    Set pt = o
    Exit For
  End If
Next

If pt Is Nothing Then
  Executive.LogHeader "No PartsTest Found"
  Exit Sub
End If

' For each part in the part
For Each pd In pt.saveagd.Test
  If pd.ComponentType = ctCAPACITANCE Then
    If pd.UseCalibration = True Then
      Set pd.Calibration = New Collection
For i = 1 To pd.SourceChannel.Count
    CMeas c, pd.SourceChannel.Item(i), pd.MeasureChannel.Item(i), _
    0.1, 0.3, 0.001, 0.00000000005
    pd.Calibration.Add c
    s = Format(pd.Reference, "!@@@@@@@@@@") & _
    " S=" & CStr(pd.SourceChannel.Item(i)) & _
    " M=" & CStr(pd.MeasureChannel.Item(i)) & _
    " Calibration=" & DoubleToEngNotation(pd.Calibration.Item(i))
    Executive.Log s, IPASS
Next i
End If
End If
Next
End Sub
Sample Test Techniques
Appendix E - Sample Test Techniques

Diode Bulk Resistance Measurements

This technique connects one side of the DUT (anode) to the current source and guards the other side (cathode). In two separate measurements, it sources two currents and measures the respective voltages generated using the DVM eas functional call. This technique then calculates the bulk resistance (dynamic resistance) as:

\[ BRES = \frac{V_{\text{meas1}} - V_{\text{meas2}}}{I_{\text{src1}} - I_{\text{src2}}} \]

Private Sub DiodeBulkRes()
    Dim RetVolts As Double
    Dim RetVolts1 As Double
    Dim BulkR As Double

    ' In two consecutive tests, measure the output voltages generated by 0.0mA and 15.0mA
    Call DVMeas(RetVolts, 108, 109, 0, 3, 0.001, 0.4)
    Call DVMeas(RetVolts1, 108, 109, 0.015, 3, 0.001, 0.4)

    ' Calculate the bulk resistance
    BulkR = (RetVolts1 - RetVolts)/0.015

    ' Check for pass/fail and display results
End Sub

Note:

Current range is limited to 50 milliamps

Measured voltage with this technique is limited to between 5 and 35 volts.

All guarded components will be put in parallel with the DUT because one side of the DUT is grounded.
FET N-Channel, P-Channel, Enhancement Mode or Depletion Mode Open Resistance and Closed Resistance Measurement

If the DUT is a depletion mode FET:

- **Note:** All JFET devices are depletion mode because they can never be used in a forward bias state.
- Test the closed FET resistance with the standard RMeas two-pin component test.
- Route the engineer’s choice of voltage source to the current source bus via SMux and connect to the gate of the FET for biasing.
- Reverse bias the gate-source to cut-off and test the open FET resistance with the standard RMeas two-pin component test.

If the DUT is an enhancement mode FET:

- **Note:** Depletion mode MOSFET devices are rare but do exist. (n-channel GaAs)
- Test the open FET resistance with the standard RMeas two-pin component test.
- Reverse bias the gate-source to activate drain-source conductivity and test the closed FET resistance with the standard RMeas two-pin component test.

An external voltage source is required to bias the devices. The source must be routed to a SMux input on bus 2 (the current source bus which would otherwise not be used during this test.) The example code uses a DAC (right pin of row 1 of the ASB slot) for the voltage source, routed to SMux input 0 of bus number 2 (left pin of row 18 of the ICAM slot.)

```
Private Sub FET_N_Depletion()

Dim RetRes As Double
Dim RetRes1 As Double

' Measure closed FET resistance
Call RMEAS(RetRes, 94, 109, 0.2, 0.3, 0.001, 167)

```
Appendix E - Sample Test Techniques

Route the DA through SMux input 0 to Bus 2 (I Source Bus)

Call SMux(0,2,95)

Set the DA voltage to open FET

Call DA(0, -1)

Measure open FET resistance

Call TwoSrcMeas(RetRes1, 94, 109, 0.2, 0.3, 0.001, 157000000)

Call DA(0,0)‘ Reset DA

‘Reset SMux channels on bus 2

Call SMux(-1, 2, 0)

‘Check for pass/fail and display results

End Sub

Private Sub FET_P_Depletion()

Dim RetRes As Double

Dim RetRes1 As Double

‘Measure closed FET resistance

Call RMeas(RetRes, 92, 108, 0.2, 0.3, 0.001, 167)

‘Route the DA through SMux input 0 to Bus 2 (I Source Bus)

Call SMux(0, 2, 95)

‘Set DA voltage to open FET

Call DA(0, 1)

‘Measure open FET resistance

Call TwoSrcMeas(RetRes1, 92, 108, 0.2, 0.3, 0.001, 157000000)

‘Reset DA

Call DA(0, 0)

‘Reset SMux channels on bus 2

Call SMux(-1, 2, 0)

‘Check for pass/fail and display results

End Sub
Private Sub FET_N_Enhancement()
Dim RetRes As Double
Dim ResRes1 As Double
'Measure opened FET resistance
Call RMeas(Retres, 36, 49, 0.2, 0.3, 0.001, 157000000)
'Route DA through SMux input 0 to Bus 2 (I Source Bus)
Call SMux(0, 2, 47)
'Set DA voltage to close FET
Call DA(0, -1)
'Measure closed FET resistance
Call TwoSrcMeas(RetRes1, 36, 49, 0.02, 0.3, 0.005, 1)
'Reset DA
Call DA(0, 0)
'Reset SMux channels on bus 2
Call SMux(-1, 2, 0)
'Check for pass/fail and display results
End Sub

Private Sub FET_P_Enhancement()
Dim RetRes As Double
Dim RetRes1 As Double
'Measure open FET resistance
Call RMeas(RetRes, 17, 36, 0.2, 0.3, 0.001, 157000000)
'Route DA through SMux input 0 to Bus 2 (I Source bus)
Call SMux(0, 2, 35)
'Set DA voltage to close FET
Call DA(0, 1)
'Measure closed FET resistance
Call TwoSrcMeas(RetRes1, 17, 36, 0.02, 0.3, 0.005, 1)
Appendix E - Sample Test Techniques

Series 2040 Test System

ICAM User Manual V3.0

258

Series 2040 Test System

‘Reset DA

Call DA(0, 0)

‘Reset SMux channels on Bus 2

Call SMux(-1, 2, 0)

‘Check for pass/fail and display results

End Sub

Notes:

The D/A converter of the Analog Source Board is limited to 16 volts @ 100mA to apply to the gate.

The gate voltage may not exceed 200 volts.

The gate current may not exceed 1 Amp.

The open FET resistance may read back negative because the current reading is well below the noise floor.

The closed FET resistance loses accuracy below 1/2 ohm because the measured current is too high with a minimum accurate forcing function of 10 mV.
Relay Coil Resistance, Open Contact Resistance, and Closed Contact Resistance Measurements

This technique will test coil resistance using the RMeas standard two-pin component test. It will:

- Test the open contact resistance using the RMeas standard two-pin component test.
- Test the closed contact resistance as follows:
  - Route an external voltage source to a SMux channel input to activate the relay coil.
  - Guard the other side of the relay coil for a ground return path.
  - Take a four wire resistance measurement of the closed contact resistance.

An external voltage source is required for the closed contact resistance test. The source must be routed to a SMux input on bus 2 (the current source bus which would otherwise not be used in this test.)

Private Sub RelayTest()
  Dim RetRes As Double
  Dim RetRes1 As Double
  Dim RetRes2 As Double

  ' Measure the coil resistance
  Call RMeas(RetRes, 192, 197, 0.2, 0.3, 0.001, 500)

  ' Measure the open contact resistance
  Call RMeas(RetRes1, 188, 201, 0.2, 0.3, 0.001, 157000000)

  ' Set the source and measurement Kelvin channels
  Call SKelvin(318)
  Call MKelvin(319)

  ' Route the DA through SMux input 0 to Bus 2 (I Source bus)
  Call SMux(0, 2, 192)
'Ground the other side of the coil

Call Guard(197)

'Set the D/A to activate the coil

Call DA(0, 5)

'Measure the closed contact resistance

Call TwoSrcMeas(RetRes2, 188, 201, 0.02, 0.3, 0.005, 1)

'Reset the source and measurement Kelvin channels

Call SKelvin(-1)

Call MKelvin(-1)

'Reset the D/A (First to avoid Inductive Kick)

Call DA(0, 0)

'Reset the SMux channels

Call SMux(-1, 2, 0)

'Remove guard from negative side of coil

Call UnGuard(197)

'Check for pass/fail and display the results

End Sub
**SCR Forward Bias Voltage Drop Test**

This technique will measure the forward-biased voltage drop of an SCR as follows:

- Route an external voltage source to a SMux input to gate the SCR.
- Apply a current to the anode of the SCR.
- Guard the cathode of the SCR.
- Measure the voltage developed across the SCR.

An external voltage source is required and must be routed to a SMux input on bus 1 (the voltage source bus which would otherwise not be used in this test.)

The example code uses DA 0 (right pin of row 1 of the ASB slot) for the voltage source, routed to SMux input 0 of bus number 1 (left pin of row 16 of the ICAM slot.)

```vba
Private Sub SCRTest()
  Dim RetVolts As Double
  'Matrix Relay channel for the SCR gate
  Dim Gate As Integer
  'Route the DA through SMux input 0 of the VSource Bus (bus 1)
  Call SMux(0,1,Gate)
  'Set the D/A voltage to gate the SCR with 1 volt
  Call DA(0,1)
  'Measure the forward-biased voltage drop of the SCR with the Source on channel 0, guard channel 1, set the current source to 20 mA. Set the current source compliance voltage to 2 volts and test in 1mS. Expect about 0.7 volts to be measured.
  Call TwoSrcDVMeas(RetVolts, 0, 1, 0.02, 2, 0.001, 0.7)
  'Reset DA
  Call DA(0, 0)
  'Reset all bus 1 SMux inputs
  Call SMux(-1, 1, 0)
End Sub
```
## Appendix F - Digalog Error Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>101:001</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:002</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:003</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:004</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:005</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:006</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:007</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:008</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:009</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:010</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:011</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:012</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:013</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:014</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:031</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:032</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:033</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:034</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:035</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:036</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:037</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:038</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:039</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:040</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:041</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:042</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:043</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:044</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:045</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:046</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:047</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:048</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:049</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:050</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:051</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:052</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:053</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:054</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:055</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:056</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:057</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:058</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:059</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:060</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:061</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:062</td>
<td>ICAM</td>
</tr>
<tr>
<td>101:063</td>
<td>ICAM</td>
</tr>
</tbody>
</table>
Glossary Of Terms
.bas File - Standard modules (.bas filename extension) contain public or module level declarations of types, constants, variables, external and public procedures.

API (Application Programmer's Interface) - Collection of Dynamic Link Libraries (DLLs) which Windows applications use to display windows, graphics, manage memory, etc.

BON (Bed Of Nails) - Test scheme where a circuit board is placed on a fixture containing probes (nails) at strategic locations (nets) such that the components on the circuit board can be verified and tested.

Class - A class module is similar to a Form module but has no visible user interface. A Class module can be used to create objects, and can contain code for defining the object.

Dialog - A custom form containing command buttons, option buttons, text boxes, etc. that allow the user to supply information to the application.

DLL (Dynamic Link Libraries) - Libraries of procedures that applications can link to at “run time” rather than be included in compiled code.

DUT - Device Under Test - The device or unit being tested. (Same as UUT)

Fixture - Hardware interface between the UUT (Unit Under Test) or DUT (Device Under Test) and the Patchboard.

Forcing Function - Type of function used as a source for a specific test. For example, the Resistance test uses a constant voltage as a “Forcing Function”. Besides a constant voltage, other forcing functions include constant current, a voltage ramp, and a voltage sinewave.

Forcing Value - Value of the source voltage, current, or frequency used in a test.

Form Module - Text description of a form and its properties.

Functional Call - Digalog subroutines written in the C programming language for the purpose of communicating with and controlling the Testhead circuitry.

GPIB - General Purpose Interface Bus - A communication bus using the IEEE-488 standard.

Guard - A ground applied at a strategic point in a circuit to isolate a component to be tested.
ICAM - In Circuit Analog Measurement. The acronym used for the ICAM card. This card is used as the measurement system for the in-circuit portion of the 2040 tester. It has features that allow the functional and in-circuit test capabilities to co-exist on the tester.

ICT - In Circuit Test. A method of testing in which the components on a circuit board are verified. As they are on the circuit board, the components are “in circuit”.

Iterator - A Digalog software technique to determine the test parameters to use for a specific part for the best standard deviation.

Kelvin (4-wire) Measurement - Use of 2 additional high impedance leads (in addition to the normal source and measure leads) to more accurately measure low resistance components.

Matrix Relay Channel - One of sixty-four channels on a Matrix Relay board which can be tied to a source, measure, or guard bus on the ICAM board.

MDA - Manufacturing Defects Analyzers

Net - A junction of two or more components in a circuit. Some test schemes also designate a net at an input or output to/from a complete circuit for testing purposes.

Open Pin Check - Checks every enabled net in the system against every other net in the system for a resistance value greater than a calculated value or a predetermined default. If the measured resistance is greater than the designated threshold, an open is said to exist.

Panelization - Ability of a test scheme to test multiple boards of the exact same type on the same fixture sequentially.

Patchboard - Digalog interface from the Testhead circuitry to the user’s fixture.

Pogo - A pin on a specific part on a net where a probe from the fixture will be placed or located.

Shorts Test - Checks every enabled net in the system against every other net in the system for a resistance value less than a calculated value or a predetermined default. If the measured resistance is less than the designated threshold, a short is said to exist.
Appendix G - Glossary of Terms

Testhead - A card cage containing all of the test circuitry in the tester including a direct hardware interface to the Patchboard. No power supplies are included in the Testhead.

UUT - Unit Under Test - The unit or device being tested. (Same as DUT)

Wirelist - Output file from AutoGen used to construct and debug a fixture.