

Chapter 6

Extensible Test Format (ETF)

Description of support for the Extensible Test Format

Introduction

The ScanExpress Runner software contains a run-time interpreter to parse and execute Extensible Test Format files. The Extensible Test Format is a simple language that allows the execution of commands outside the normal ScanExpress Runner Boundary-Scan environment. Using ETF makes it possible to control external test equipment such as relay controllers, and digital multi meters. ETF has the ability to execute external Windows and DOS programs from within ScanExpress Runner. It is also possible to capture the output of commands, and display it in a pop-up window during test execution. You can add delays to allow time for external circuitry to initialize and settle. It is even possible to insert documentation into the ETF file using the '#' comment command.

ETF files are created in standard ASCII text format using a program like notepad. ETF files are added into the test plan, just as you would add a CVF, SVF or FPI file. To ScanExpress Runner the ETF file appears to be a single test step, but inside the ETF file can be many individual commands. This allows for a very convenient method of partitioning test steps into logically related sections. For example, if you are performing a variety of analog tests you can place all of the setup initialization, test procedure, and posttest cleanup all in a single ETF file. Or, you may choose to break it up so that each test has its own ETF file, with all of the specific initialization code for that particular test encapsulated. It is even possible to embed CVF files inside an ETF file. In this way you have the combined power of controlling external instruments, initializing circuitry, and then performing test or control methods using boundary-scan technology.

The ETF command parser understands the following commands:

CVF, RUN, SET PIO, SET DMM, READ PIO, READ DMM, WAIT PIO, DELAY, PRINT, COMPARE, POWERSHORT and # (for comment)

Sample ETF File

See Figure 6-1 for a sample ETF file.

This ETF file is used to measure the current draw in a circuit. The digital multi-meter is placed in DC current mode, using the 3.3mA range. The circuit is configured using an external relay controller, and a stimulus is applied to the circuit using Boundary-Scan control registers. The tester waits 500 milliseconds and takes a current reading, saving it in the variable VALUE1. A different stimulus is applied; the tester waits 100 milliseconds, and takes another current reading. The two values are compared, and if the difference is +/- 4 micro amps the test is failed.

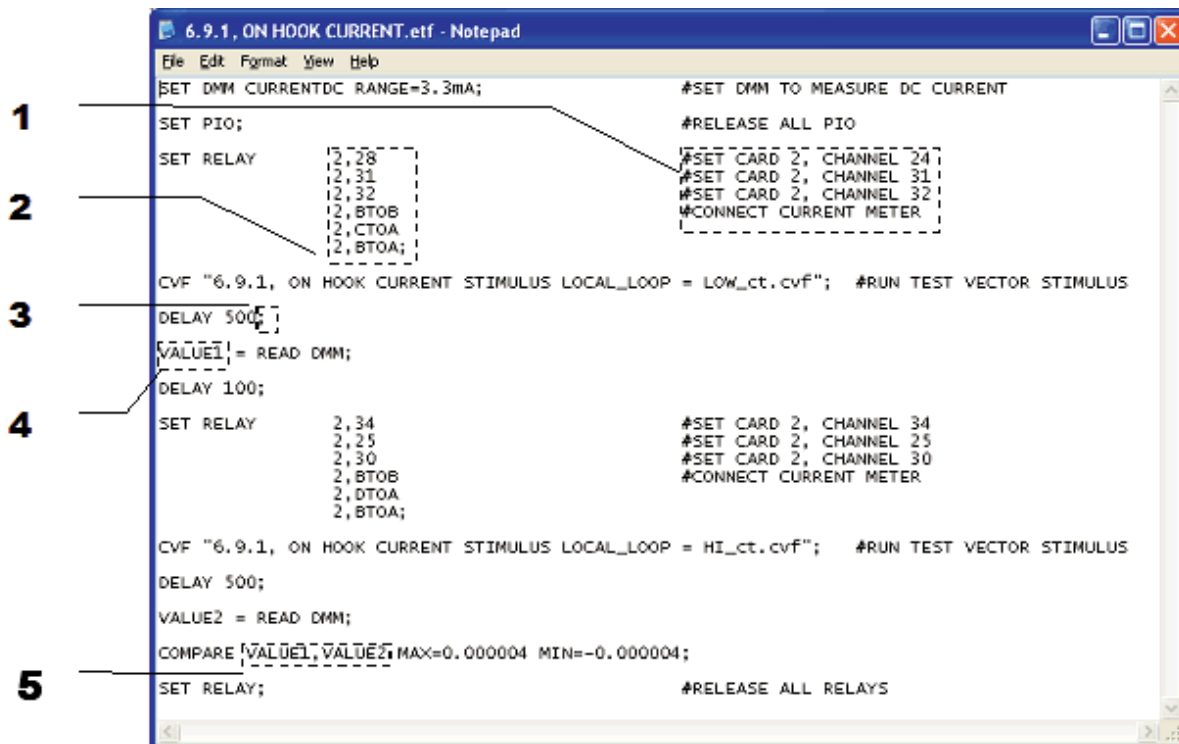


Figure 6-1. Sample ETF file

1. Comments help document the test file.

A comment consists of the '#' character and all of the following characters up to the end of the line. Comments are ignored during processing of the ETF file.

2. A command may be split over multiple lines.

In the example above, the SET RELAY command is spread over several lines. This makes it easier to see which relays are being set. It would also be perfectly valid to place all of the command on one line as follows:

```
SET RELAY 2,28,2,31,2,32,2,BTOB,2,CTOA,2,BTOA;
```

3. Each command must end with a semi-colon: ";".

The semi-colon designates the end of a command. This allows complex commands to be split over multiple lines, as described in step 2 above.

4. Output from the READ and COMPARE command may be saved into a variable.

As can be seen from above, the results of reading the digital multi-meter are saved to a variable named VALUE1. Variables can hold string or numeric data.

Examples: ExpectedVoltage = 12.0V;

MyOhms = 33k;

A\$ = "This is a string variable";

You can also capture the result of a READ DMM, READ PIO or COMPARE command with a variable.

```
Volts1 = READ DMM;  
Volts2 = READ DMM;  
DeltaVolts = COMPARE Volts1, Volts2;
```

5. The COMPARE and PRINT commands may use a variable as an argument.

Sample Variable Usage

Figure 6-3 below shows an example of using variables to capture the input signals connected to the PIO connectors on the PCI JTAG controller. This test drives bit A4 low using the SET PIO !A4 command. It then waits up to 3 seconds for PIO bit B5 to be high. In the example below this condition succeeds, and then the A4 bit is driven high. The tester will wait up to 3 seconds (the specified time out value) for the B5 bit to go low. As can be seen below, this does not happen within the 3-second time limit, and the WAIT PIO command issues a failure notice. The detailed display shows both the expected and actual values.

At this point the test has failed, but execution will continue to the end of the ETF file. This allows any post-test cleanup to take place. The results of the test are passed back to the ScanExpress Runner main program. (See Figure 6-2 below)

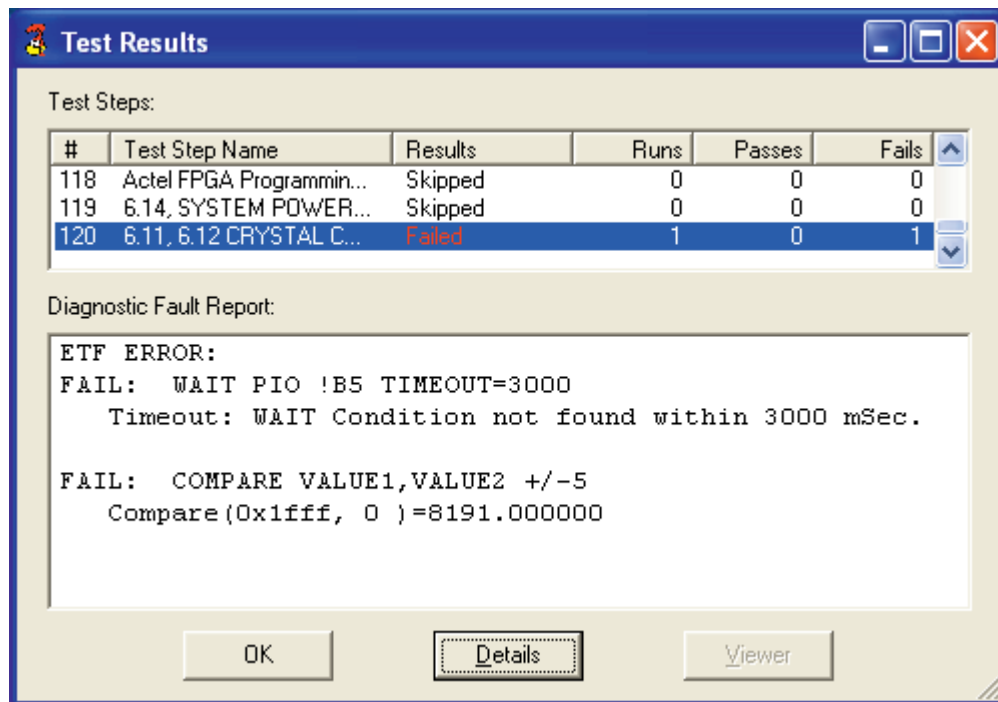


Figure 6-2. WAIT PIO timeout results

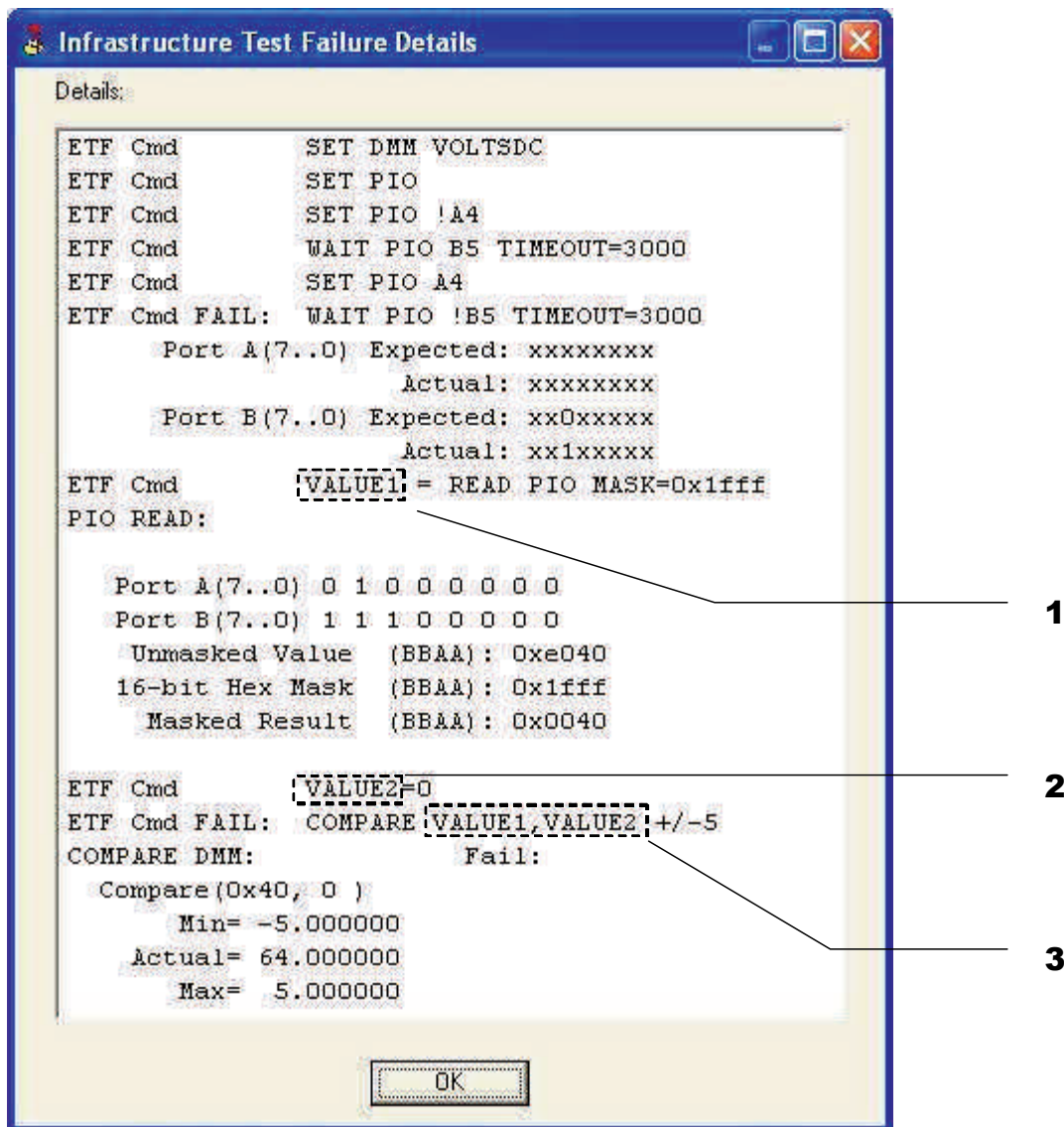


Figure 6-3. ETF Program Variables

1. Variable assigned from READ PIO

The variable VALUE1 is assigned the results of the READ PIO command. This value will be used later in a COMPARE command to verify the contents of VALUE1 fall within a specified range. In this particular case the Parallel I/O bits of the PCI JTAG controller are connected to an external device. The READ PIO command reads the state of these bits, and returns the value as a numerical value.

2. Variable assigned a constant:

The variable VALUE2 is assigned the value of zero. This value will be used as an argument to the COMPARE command.

3. Variables as arguments to the COMPARE command:

The variable VALUE1 and VALUE2 are provided as arguments to the COMPARE command. COMPARE will subtract VALUE2 from VALUE1 and compare it against the specified range. If the result falls within the specified range the COMPARE command will return a “pass” condition, otherwise it will return a “fail” condition.

Sample PRINT Command

It is possible to pause a test and display a message in a pop-up window (see Figure 6-4 below). Execution will halt until the pop-up window is closed. This is accomplished using the PRINT command.

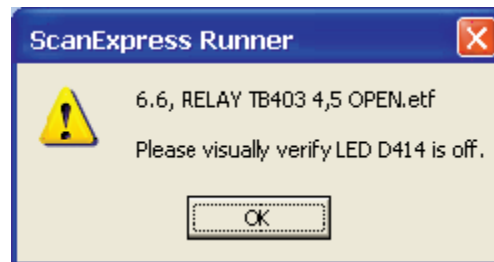


Figure 6-4. PRINT pop-up window

In Figure 6-5 below the PRINT command activates a pop-up window with the message “Please visually verify LED D414 is off.” Test execution will halt until the user closes the window by clicking on ‘OK’ or the ‘X’ (See Figure 6-4).

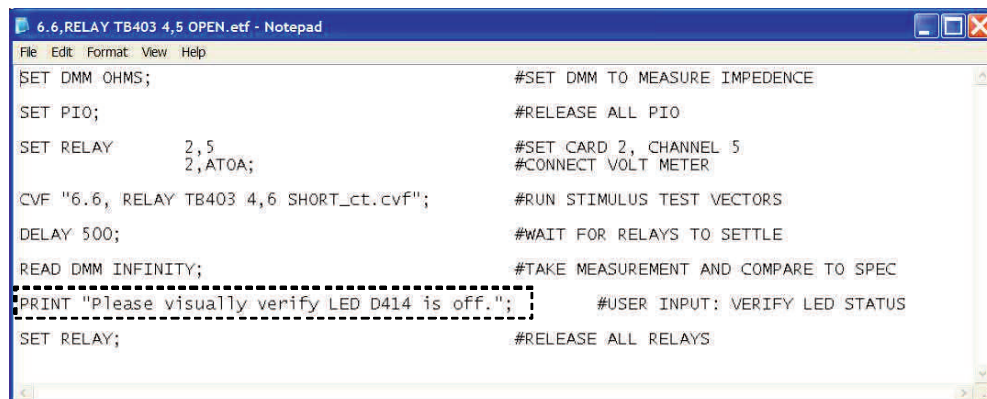


Figure 6-5. Sample PRINT command in ETF file

It is also possible to print the contents of a variable.

Example: MyMessage\$ = "This is a message that will print in a Windows Message Box";
 PRINT MyMessage\$;

PASS or FAIL Test Results

ETF files (test steps) return a result code (PASS or FAIL) similar to the other types of test steps. An ETF file is considered to pass if every command within it executes successfully and without error. Conversely, the ETF test step is considered to fail if any one or more of the commands within the ETF file fail. See Figure 6-6 for an example of an ETF test step reporting a failure to ScanExpress Runner.

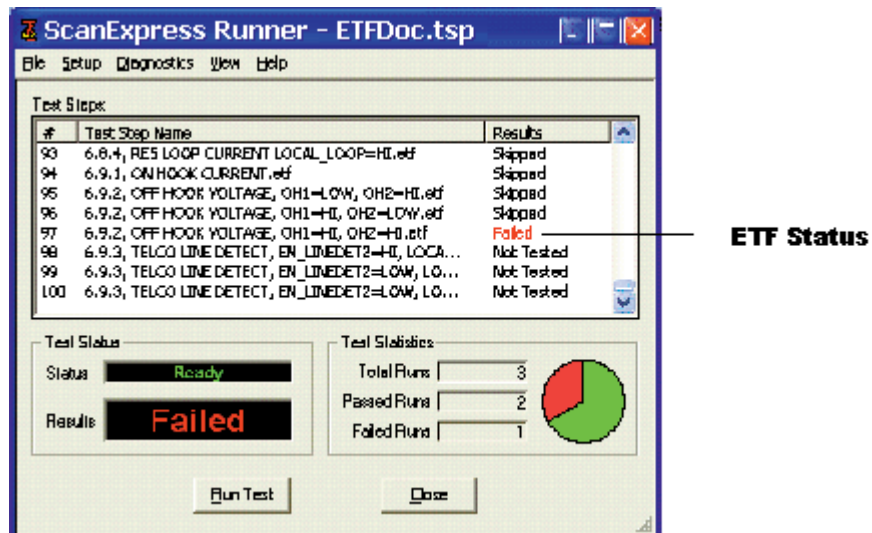


Figure 6-6. ETF test step results: Main Window

The COMPARE and READ commands have the ability to PASS/FAIL a test based on an expected value and tolerance, or a min and max value. The RUN command returns a PASS/FAIL based on the DOS Error Level or Windows Return Code generated during execution. Additionally the CVF command has the ability to PASS/FAIL a test based on the results of the boundary-scan operations.

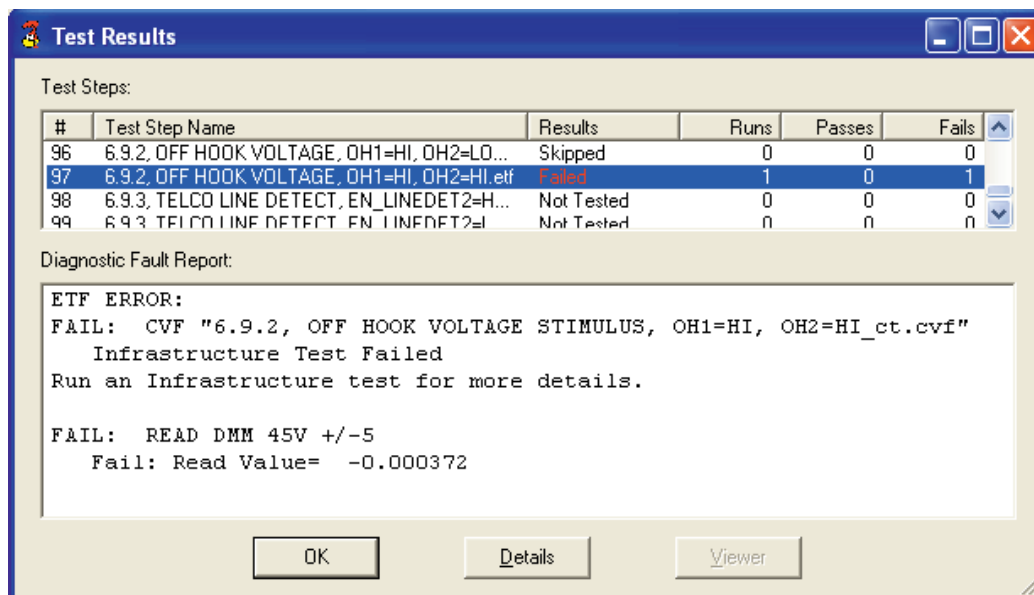


Figure 6-7. ETF test step Result Dialog

These results are passed back to the main ScanExpress Runner executive. This allows the ETF file to act just like any other test step in a test plan. As with any other test step, the results dialog will

contain status information for any selected test step. If the test step was a success, the Diagnostic Fault Report will be empty. If there were one or more failures in the ETF test step, the Diagnostic Fault Report will contain a description of each failure. See Figure 6-7 for the example Results Dialog that matches the main screen shown in Figure 6-6 above.

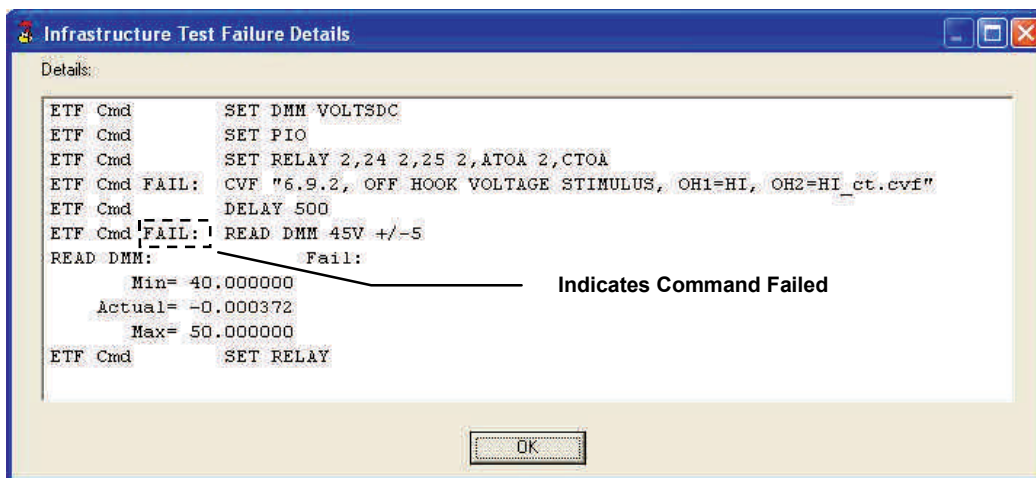


Figure 6-8. ETF test step failed: Details Dialog

Detailed information is gathered during execution of an ETF test step. This detailed information is available regardless of the pass/fail condition of the ETF file. At a minimum, the detailed information will contain an entry for each command executed in the ETF file. Comments and blank lines are ignored, and not reported. Figure 6-8 refers to an ETF file with failures. This is the same test step depicted in Figure 6-6 and Figure 6-7 above. Figure 6-9 is an example of an ETF test step that executed without failure.

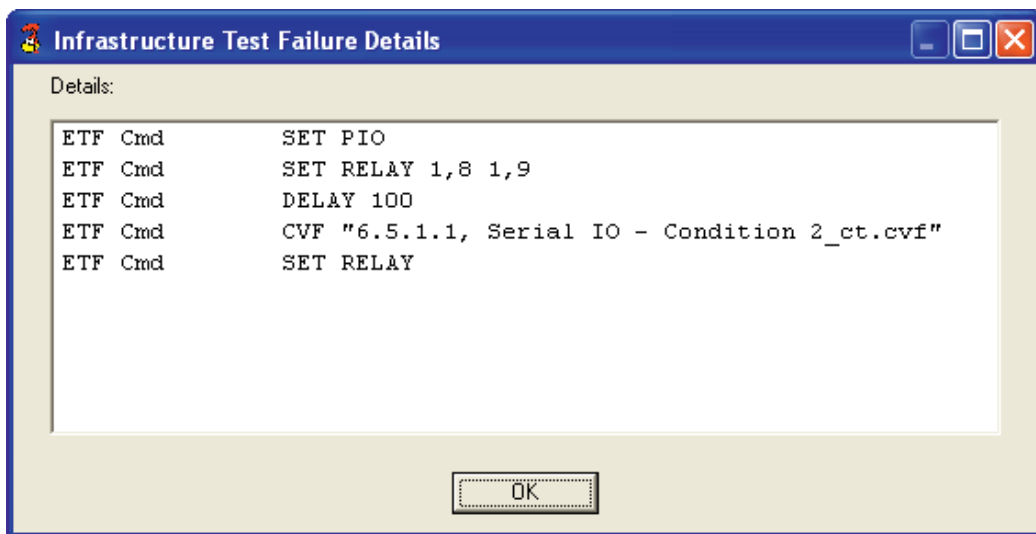


Figure 6-9. ETF test step passed: Details Dialog

Ranges, Tolerances, and Numerical Suffixes

Numerical values may use the following suffixes:

mV, mA, K, MEG

Where mV = millivolts, mA=milliamps, K=1,000 and MEG = 1,000,000.

The parser will convert any numeric value using one of the suffixes to a normalized value for internal use. Normalized values are represented internally as double precision floating point values. It is possible to freely mix normalized values and numerical values with suffixes in the same operation. Refer to Figure 6-10 for an example of a numerical suffix.

250mV will be converted to 0.25

22.415K will be converted to 22415.0

1.7152MEG will be converted to 1715200.0

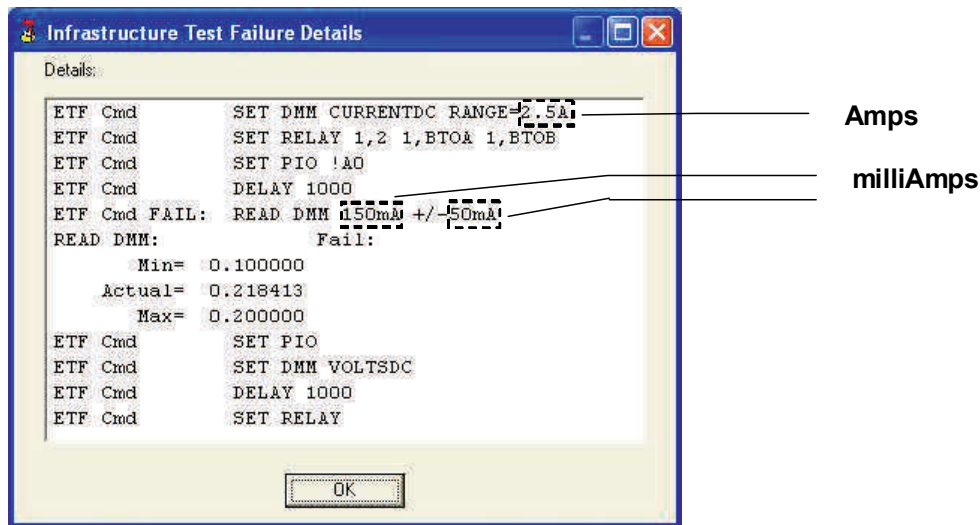


Figure 6-10. Example Numerical Suffix

A range consists of a Minimum and a Maximum value. A range is specified by entering a MAX=value, MIN=value as part of the READ DMM command. *Value* must be a numeric value.

The special symbol "+/-" *numeric* is used when entering a tolerance. A tolerance is used when comparing against an expected value. A tolerance is converted internally to a range (minimum and maximum value) using the expected value by setting MIN=expected value - *numeric*, and MAX=expected value + *numeric*. The special symbol "%" will cause the tolerance to be used as a percent of the expected value. See Figure 6-11 for an example of using a tolerance with an expected value.

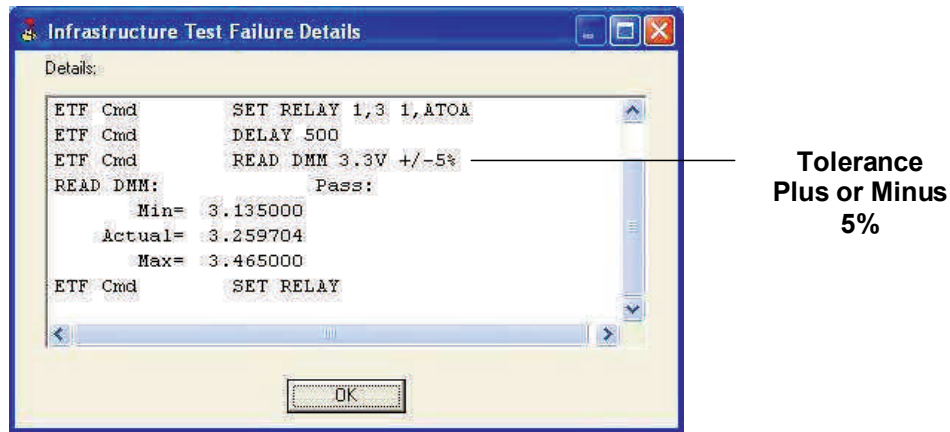


Figure 6-11. Example Tolerance (per cent)

Refer to Figure 6-11 for the following description:

The expected value is 3.3 volts. The tolerance is +/- 5 per cent.

Min is calculated as $3.3 - 10\%$ of 3.3, or 3.135.

Max is calculated as $3.3 + 10\%$ of 3.3, or 3.465.

The actual value read back from the DMM was 3.259704 volts. This passes the test because 3.259704 volts is between the minimum and maximum values, or $3.135 \leq 3.259704 \leq 3.465$.

It is possible to use a MIN and MAX value wherever a tolerance is expected. When explicitly setting a MIN and MAX value it is not necessary to provide an expected value. Refer to In Figure 6-12 for the following description. It would be perfectly valid to replace the "26mA +/- 1mA" with "MIN=25mA, MAX=27mA" or "MIN=0.025, MAX=0.027".

The expected value is 26 milliamps. The allowable tolerance is +/- 1 milliamp.

Min is calculated as $26 - 1$, normalized into amps, or 0.025. (25 milliamps)

Max is calculated as $26 + 1$, normalized into amps, or 0.027. (27 milliamps)

The actual value read back from the DMM was 0.02575 (25.75 milliamps). This passes the test because 25.75 mA is greater than 25 mA, and less than 27 mA.

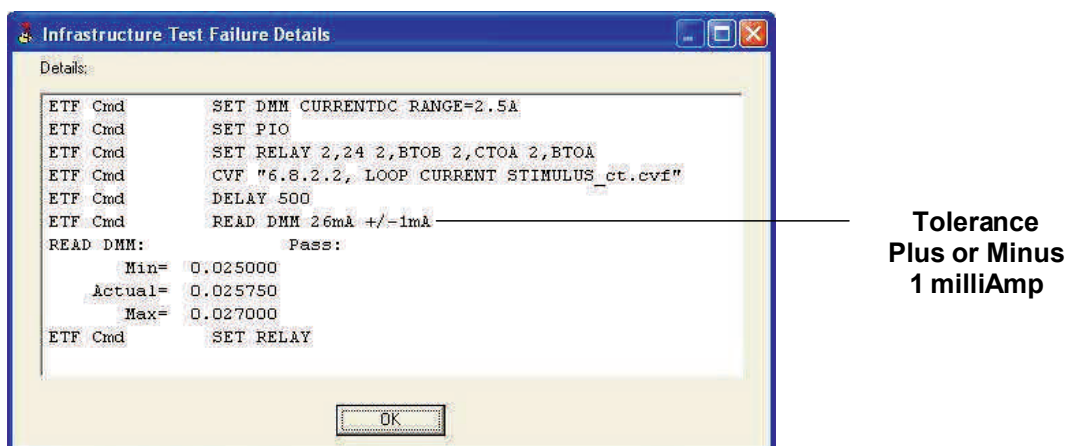


Figure 6-12. Example Tolerance

COMPARE Command

The COMPARE command is used to test the results from two separate commands. This is slightly different from the READ DMM command, which can optionally perform a comparison against an expected value. The COMPARE command subtracts the value of the first argument from the value of a second argument and determines if the result falls within a specified range. The range can be specified either as a MIN and MAX value, or as an expected value and a tolerance. If the comparison does not fall within the selected range, COMPARE will issue a FAIL condition to ScanExpress Runner. Additionally, the value resulting from the subtraction may be saved into a variable.

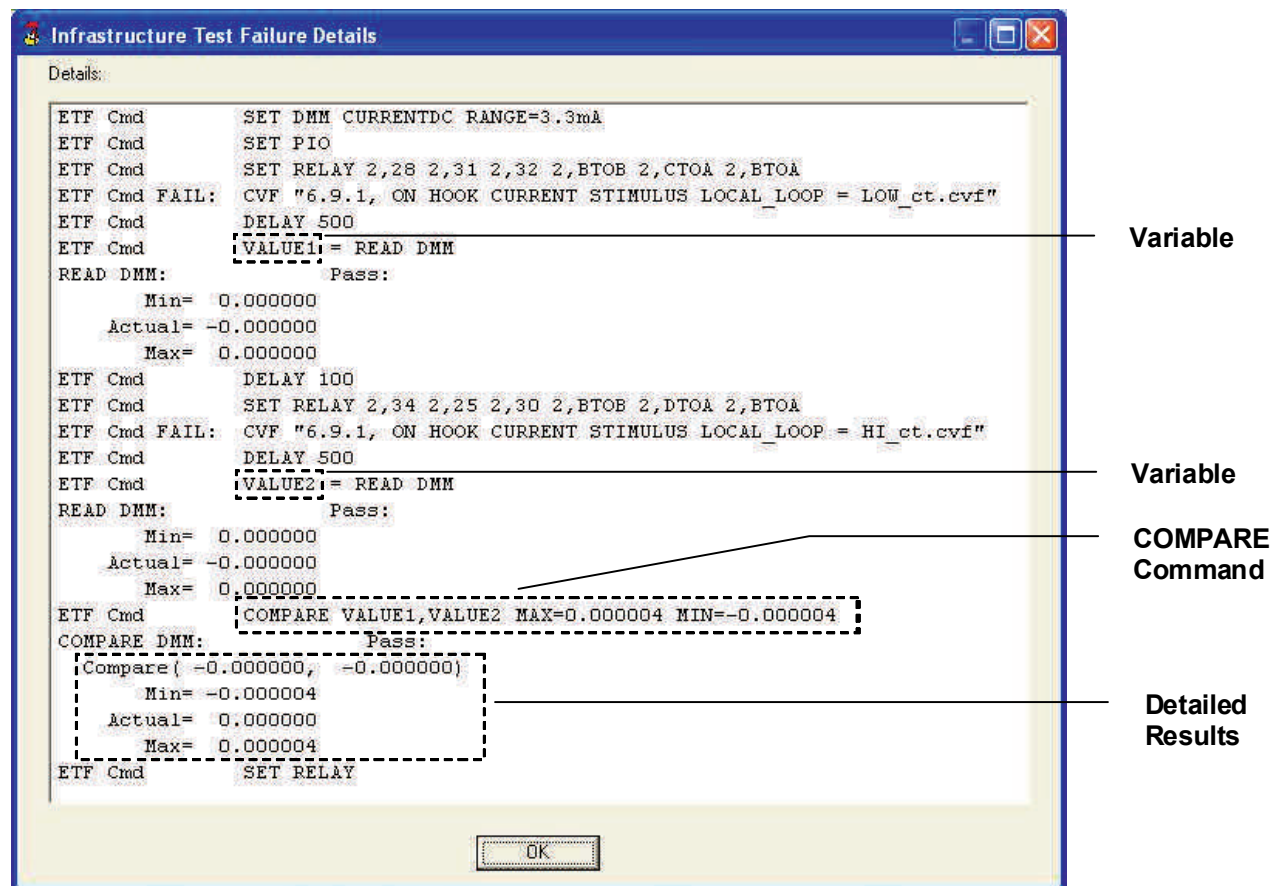


Figure 6-13. Example COMPARE Command

Refer to Figure 6-13 for the following description of the COMPARE command:

The digital multi-meter is placed into the DC current mode using the 3.3-milliamp range. The PIO is placed in tri-state mode, and setting the appropriate relays configures the circuit. The CVF command applies the necessary stimulus, and the test pauses for 500 milliseconds to allow the circuit time to settle. The DMM measures the current passing through the circuit under test, and saves the value into a variable. This process is repeated for another circuit configuration, and the current measurement is again saved into a variable. For this test the current readings must be within 4 microamps of each other. This is specified by setting the range to a MAX of 0.000004 and a MIN of -0.000004.

In this particular case we can see that the input current measured by the DMM was zero for both readings, so value1 = 0, and value2 = 0. The COMPARE command subtracts value1 from value 2 with a result of zero. This happens to pass because zero is between the MIN and MAX range values. We can see from the detailed result all of the values necessary to evaluate the test condition.

A closer inspection of the details shows that the CVF command failed before the DMM could read the current values. This failure is reported to ScanExpress Runner, and reflected in the test step results. See Figure 6-14 below.

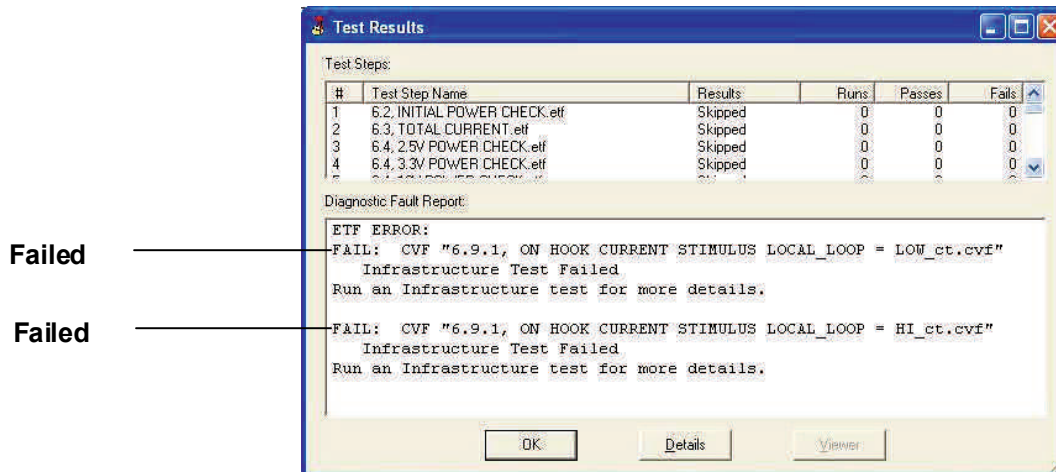


Figure 6-14. Results of failed CVF command

PIO Specific Commands

The PIO consists of two eight-bit digital I/O ports located on the PCI 1149.1 JTAG controller, or the PCI Turbo controller. These ports are user configurable, and can be used for general-purpose input and output. There are several PIO specific commands that allow you to tri-state the PIO outputs, or drive any individual pin or combinations of pins to a logical high or low.

SET PIO

The SET PIO command controls the output state of the I/O ports. SET PIO with no arguments causes both port A and port B to place their outputs into a tri-state condition. SET PIO can have multiple arguments. Each argument specifies a single pin to drive either high or low. Specifying the pin name will drive the output high. Placing an exclamation point before the signal name will have the inverse effect; it will drive the output low. Any signal that is not explicitly set to a low will default to a high condition. Specify multiple pins by separating them with a space or a comma.

Referring to Figure 6-15 below:

1. Tri-states the outputs.
2. Sets bit 4 of port A low while setting the remainder of port A and all of port B high.
3. Explicitly sets bit 4 of port A high while implicitly setting all of the other bits high as well.

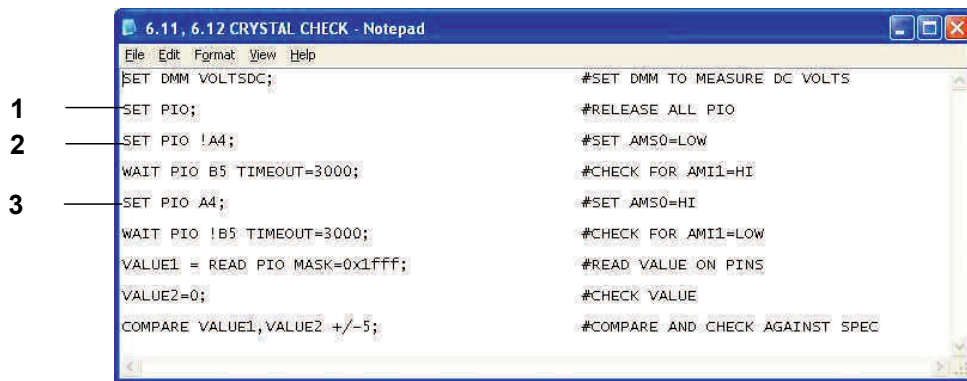


Figure 6-15. Example SET PIO command

READ PIO

The READ PIO command reads the state of the general purpose input bits of port A and port B. It constructs a 16-bit value by placing the contents of PORT A in the lower 8 bits and PORT B in the upper 8 bits. The resulting value may be assigned to a variable. An optional MASK argument will strip off all bits not specified in the mask. This allows non-connected pins to be ignored.

Figure 6-16 shows the detailed output of a READ PIO command. The command specifies a MASK of 0x1fff. This has the effect of stripping off the upper 3 bits. All bits that are masked off will have a zero value in the result. In other words, the resulting value of the READ PIO command will be in the range from zero to 0x1fff (See item 3 in Figure 6-16).

The detailed results of a READ PIO with mask can be a little confusing. Take a look at item 2 in Figure 6-16. The “Port A (7..0) 0 1 0 0 0 0 0 0” line indicates the contents of PIO port A. The value is a binary 01000000, or 0x40 in hexadecimal. The “Port B (7..0) 1 1 1 0 0 0 0 0” line indicates the contents of PIO port B. The value is a binary 11100000, or 0xE0 in hexadecimal. After combining the values for port A and port B we get the 16-bit value 0xE004. This is the “unmasked” value constructed by shifting the value for port B into the upper 8-bits of our 16-bit value. The mask we provided as an argument to the READ PIO command (0x1fff) is logically “anded” with the 16-bit unmasked value (0xE004) to produce the final result. This final result is called the “masked result”, and this is the value that is assigned to the variable (0x0040).

WAIT PIO

The WAIT PIO command (see item 1 in Figure 6-16) monitors the input bits of the PIO until a specified condition is met. If the condition is not met, the command will wait indefinitely unless a TIMEOUT value is specified. The time out value is entered in milliseconds. The wait condition is specified in a manner similar to the SET PIO command. A condition is built by explicitly listing each bit and its desired state. Any bit not specified is masked out and not significant to the WAIT command.

Item 1 of Figure 6-16 shows a very simple condition. The test waits until Bit 5 of port B goes low. If the bit does not go low within three seconds (3000 milliseconds) the WAIT command will notify ScanExpress Runner of a FAIL condition. If you examine the detailed display closely you will see the expected value, and the actual value at the conclusion of the WAIT command. Each ‘x’ in the display indicates a masked, or “don’t care” bit.

More complex conditions are possible. For example, to wait for all of the odd bits in port A to go low while all of the odd bits in port B go high you would specify the following command (don't forget to specify a timeout value!):

```
WAIT PIO !A1, !A3, !A5, !A7, B1,B3,B5,B7 TIMEOUT=2500;
```

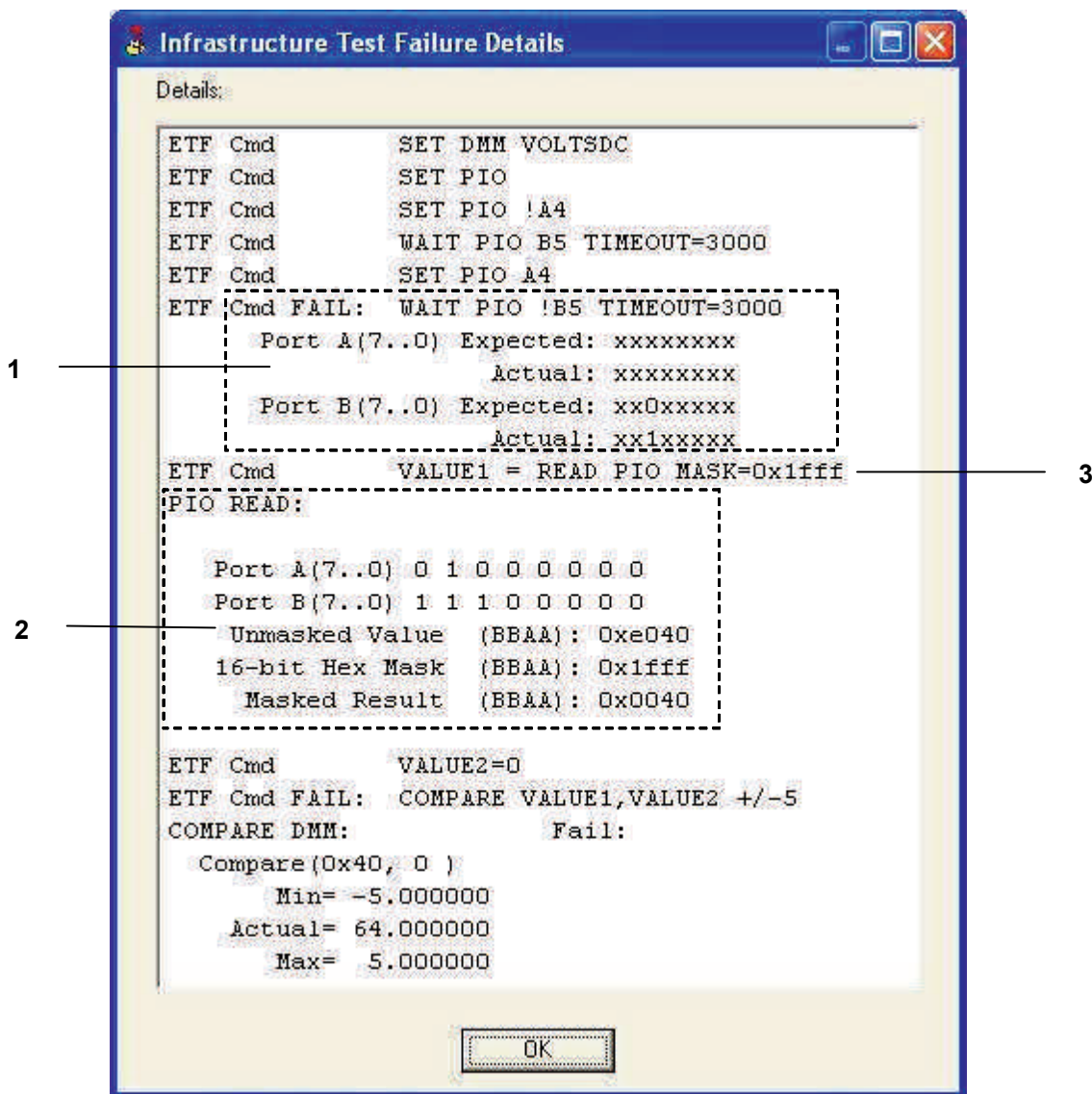


Figure 6-16. Example PIO command details

Figure 6-16 shows a detailed diagnostic display of an ETF test step utilizing the PIO commands. It starts out by configuring the digital multi-meter to read dc voltage. That is always a good safety precaution to prevent damage to the DMM, especially if it had previously been set to a very low current mode. Next, the PIO is put into tri-state. Then bit 4 of output port A is driven low. The test then pauses until bit 5 of port B goes high, or three seconds elapses. Next, bit 4 of port A is driven high. The test pauses again, until bit 5 of port B goes low, or it times out. In this particular case, bit 5 never went low, so the test reports a FAIL condition to ScanExpress Runner. After that it reads the PIO input bits, and masks off the upper three bits. It saves the resulting value into a variable named

VALUE1. It assigns the value of zero to another variable named VALUE2. It then performs a comparison by subtracting VALUE2 from VALUE1. It compares that against the tolerance specified of plus or minus 5. Because an expected value was not explicitly specified, the expected value defaults to zero. Internally the tolerance of +/- 5 is converted into a range of 0-5 and 0+5. In this case, the comparison fails because 64 (0x40) is greater than 5.

DMM Specific Commands

The DMM command controls an external digital multi-meter suitable for measuring voltage, current, and resistance. The SET DMM command selects the mode of operation. The READ DMM command performs a measurement, and optionally compares the measurement to a previously determined expected value.

SET DMM

This command is used to set the mode for the digital multi-meter. The mode consists of the *function*, *range*, and *rate*.

The *function* determines what type of measurements the DMM will make. The valid keywords for selecting the measurement mode are VOLTSAC, VOLTSDC, CURRENTAC, CURRENTDC and OHMS.

The *range* selects the measurement range of the DMM. The range is a numeric value and may include units and/or decimal modifiers. Units (such as V for volts, or A for amps) are ignored, and used only for clarity in expression. Decimal modifiers are applied to the numeric value to provide a “normalized” number. For example, 10K is equivalent to 10000.0, and 250mV is equivalent to 0.25.

The *rate* is the number of samples per second the DMM will use when programmatically taking multiple readings. Normally you will not have to set a rate value. The rate must be set to at least 1 sample per second. The default value is 10 samples per second.

Usage:

SET DMM *function* **RANGE**=*range* { **RATE**=*rate* };

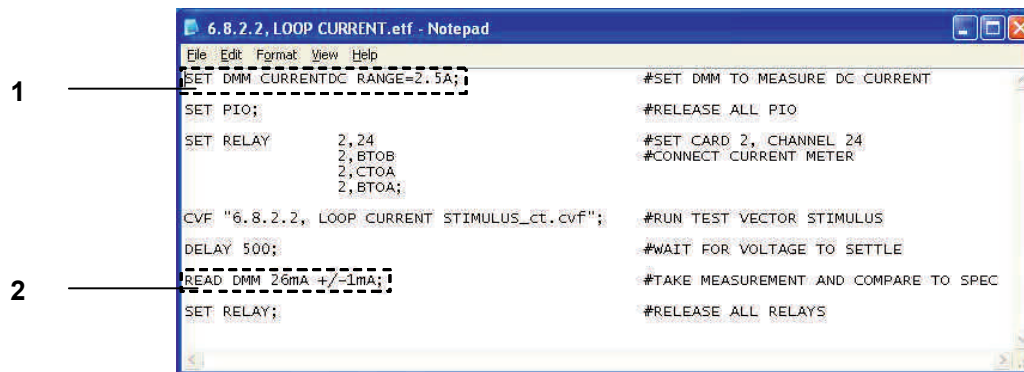


Figure 6-17. ETF file utilizing the DMM to measure DC Current

Item 1 in Figure 6-17 illustrates how to place the DMM into a mode to perform DC current measurements. The RANGE assignment selects the 2.5 amps scale.

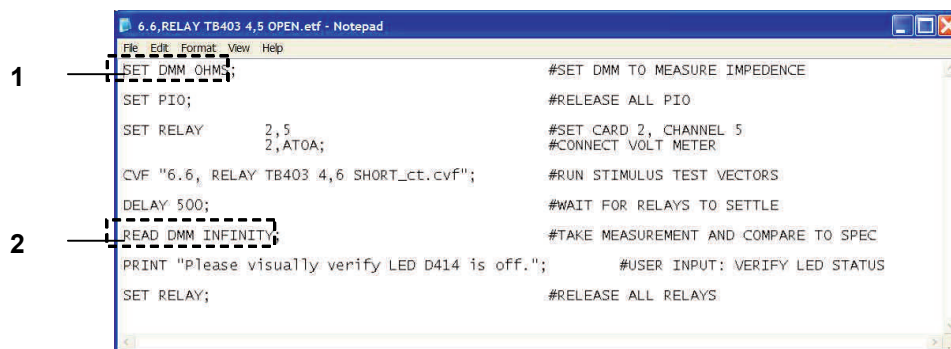


Figure 6-18. ETF file utilizing the DMM to measure Resistance

Item 1 in Figure 6-18 illustrates how to configure the DMM to perform resistance measurements. In the example above there is no RANGE assignment. It is not necessary to set a range when performing continuity checks.

READ DMM

This command causes the digital multi-meter to perform a measurement using the currently selected mode and range. The measurement may be saved into a variable. The READ DMM command can optionally perform a PASS/FAIL comparison against a previously specified range of values. The range may be specified by explicitly setting the MIN and MAX values, or it may be calculated using an expected value and a tolerance.

Sometimes a measurement may exceed the specified range of the digital multi-meter. An example of this condition would be trying to measure the resistance of an open circuit. This is a very common operation when checking the continuity of a circuit. The digital multi-meter is not able to calculate a numerical value for infinity (roughly the amount of ohms of resistance in an open circuit), so it returns an “over range” condition. The keyword READ DMM INFINITY (see Figure 6-18, Figure 6-19) provides an expected value suitable for measuring values outside the normal range.

The selected range of the DMM has a large impact on how the measurement is reported. Measuring a ten mega-ohm resistor with the range set to 1MEG results in a perfectly normal measurement. Performing the same operation with the DMM set to the 300 ohms range results in an over range condition.

Usage:

```
{var = }READ DMM {expected, tolerance} {MAX=xxx, MIN=xxx};
```

OHMS

Figure 6-19 shows the details of a failed continuity check. A more detailed explanation follows:

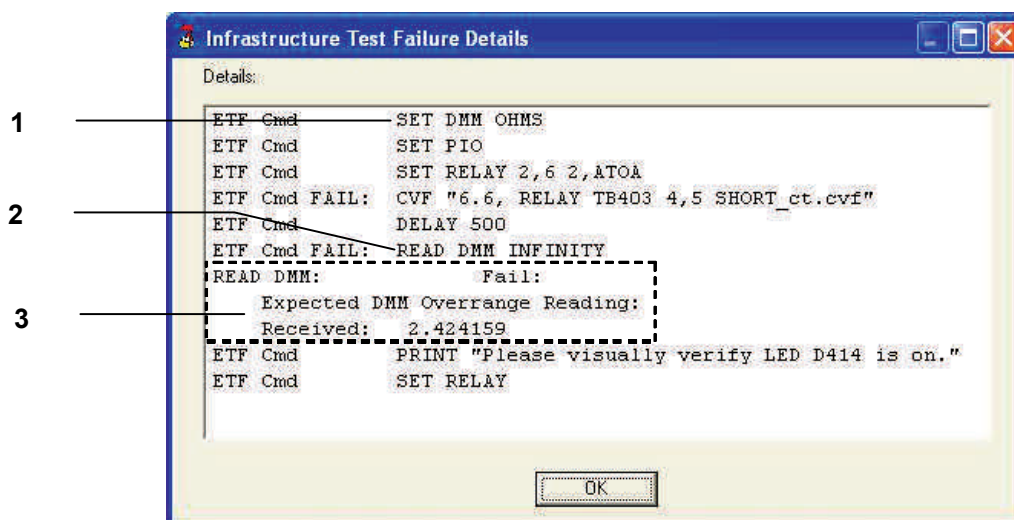


Figure 6-19. Details of a continuity check

1. SET DMM OHMS

This places the digital multi-meter into the mode to make resistance measurements.

2. READ DMM INFINITY

This will cause the DMM to perform a resistance measurement, and compare the result to an over range condition. In other words, we want to test for an open circuit.

3. Detailed Command Results

The DMM was expecting to encounter an over range condition. The actual measurement was 2.4 ohms. Instead of finding an open circuit, the DMM detects a short circuit. The FAIL condition is reported back to ScanExpress Runner.

DC Current

Figure 6-20 shows the detailed result of comparing the results of two measurements of current. The DMM is placed into the DC current mode, with a selected range of 3 milliamps. The circuitry is initialized, and the DMM takes a current reading. The result is saved in a variable. The process is repeated, and the results are stored in a second variable. The two values are compared, and the

results reported to ScanExpress Runner. The relays are then set open, removing voltage from the unit under test.

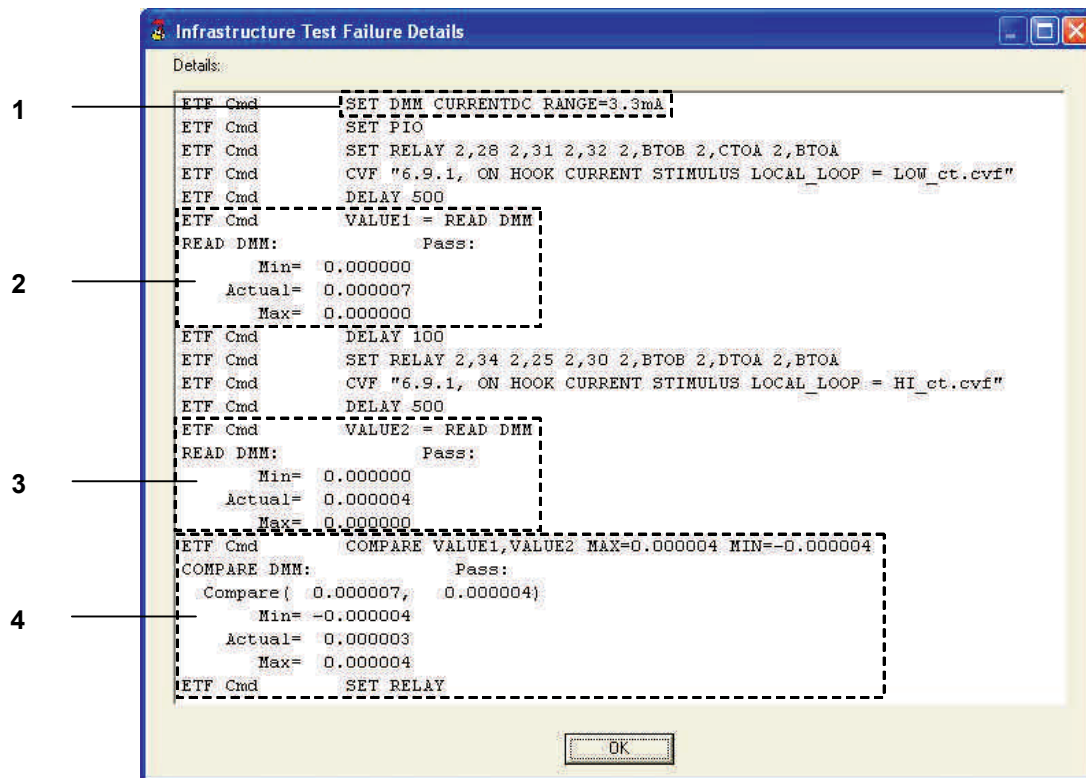


Figure 6-20. Measuring Current with the DMM

1 SET DMM CURRENTDC RANGE=3.3mA

This command places the digital multi-meter in the mode to perform measurements of direct current. The range is set to 3.3 milliamps for measuring low current circuitry.

2 VALUE1=READ DMM

This command will cause the digital multi-meter to perform a DC current measurement. The result is stored in the variable VALUE1. The detailed display window shows the actual value of the current to be 7 microamps. The result passes by default because no expected value or range was specified to test against.

3 VALUE2 = READ DMM

This command is similar to item #2 above. The value is stored into a second variable. The value read is 4 microamps.

4 COMPARE VALUE1,VALUE2 MAX=0.000004 MIN=-0.000004

This command compares the result of the two current measurements. The acceptable range is specified using MAX and MIN. The compare command subtracts the variable VALUE2 from VALUE1, obtaining a result of 0.000003. This value falls between the minimum value of -4 microamps and +4 microamps, thus the comparison is a success and is reported as such to ScanExpress Runner.

RELAY Specific Commands

The ETF parser contains built-in support for an external relay controller. The current configuration allows the user to control up to four PCI based Signametrics relay multiplexers. The controllers are designated 1,2,3,4 where 1 is the first controller PCI card and 4 is the last. The controller number is always specified in a SET RELAY command except when clearing all of the relays at once. The relays are controlled in what is called “universal mode”, which means every relay is individually selectable. Channel relays are numbered from 1 to however many are on the controller card. Typically there will be 20 or 40 channel relays per card. Configuration and Tree relays are identified by name: AtoA, BtoA, BtoB, CtoA, CtoC, DtoA, DtoC, DtoD.

Usage:

```
SET RELAY { card, relay { ,card,relay... } };
```

SET RELAY

A SET RELAY command with no arguments will release all of the relays. To activate a single relay, select the card number and relay number. Any relay not explicitly command to activate will automatically deactivate. To activate more than one relay at a time combine all of the relays into a single SET RELAY command. Refer to Figure 6-21 to see how to use the SET RELAY command.

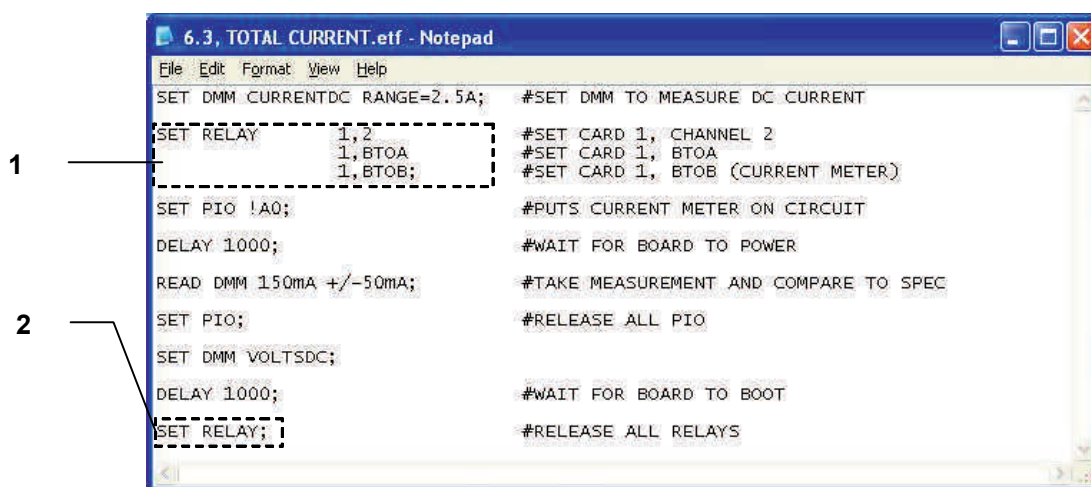


Figure 6-21. Set Relay Command

1 Setting Multiple Relays

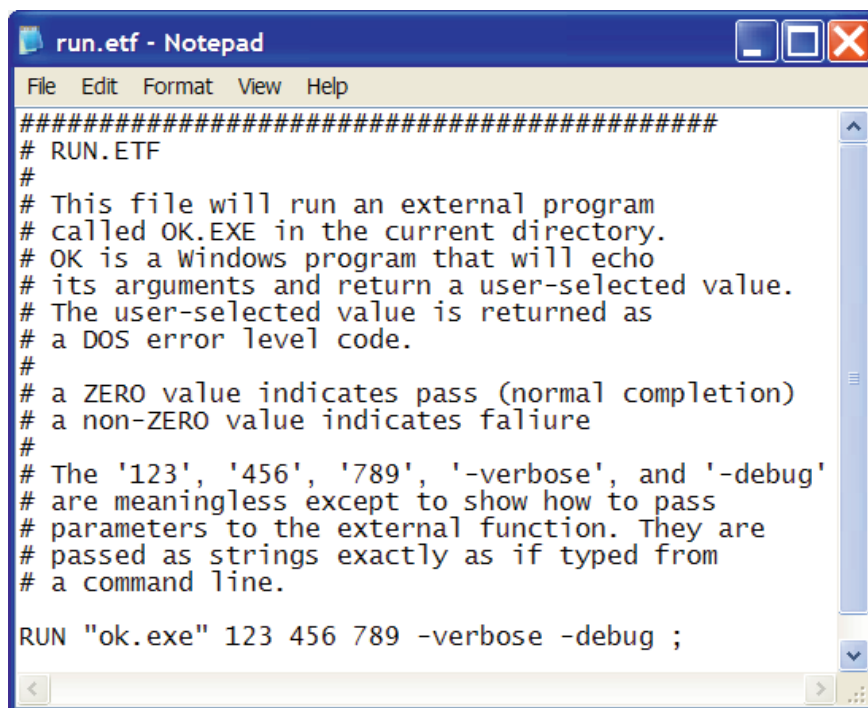
This command is setting one of the channel relays and two of the configuration relays. Notice how the command is split over several lines of the ETF file. This makes it much easier to see which relays are being set. The comments make it easier to understand what the command is doing.

2 Releasing All Relays

This command deactivates all of the relays at once.

RUN Command

The RUN command executes an external program. The program may be a Windows program, a DOS program, or a BATCH command file. Execution of the current test step will pause until the external program is complete and exits. The external program may return a value. If the value is non-zero the RUN command will report a FAIL condition to ScanExpress Runner.



```
run.etf - Notepad
File Edit Format View Help
#####
# RUN.ETF
#
# This file will run an external program
# called OK.EXE in the current directory.
# OK is a Windows program that will echo
# its arguments and return a user-selected value.
# The user-selected value is returned as
# a DOS error level code.
#
# a ZERO value indicates pass (normal completion)
# a non-ZERO value indicates failure
#
# The '123', '456', '789', '-verbose', and '-debug'
# are meaningless except to show how to pass
# parameters to the external function. They are
# passed as strings exactly as if typed from
# a command line.

RUN "ok.exe" 123 456 789 -verbose -debug ;
```

Figure 6-22. RUN Command ETF file

Figure 6-22 shows an ETF file that issues a run command. The first argument to the run command is the fully qualified path of the program to execute. The quotes around the file name are optional unless the file name or path contains spaces in which case the quotes are required. If you leave the quotes off a file name that contains spaces the parser will take everything up to the first space as the file specifier. Everything following the space will be treated as an argument. Surrounding the file name and path with quotes forces the ETF parser to treat them as a single argument.

The program C:\BIN\OK.EXE is a very simple Windows program that displays the arguments used to call it. It also returns a user-selectable value upon exit. This provides a very simple method to see the RUN command in action. Figure 6-23 shows the OK program after we called it with the RUN command. Notice that the arguments are the same as in the ETF file with one exception. The `-verbose` and `-debug` arguments are now capitalized! That is because the ETF parser is case insensitive and converts its arguments to upper case internally. The file name is still in lower case because it was surrounded by quotes.

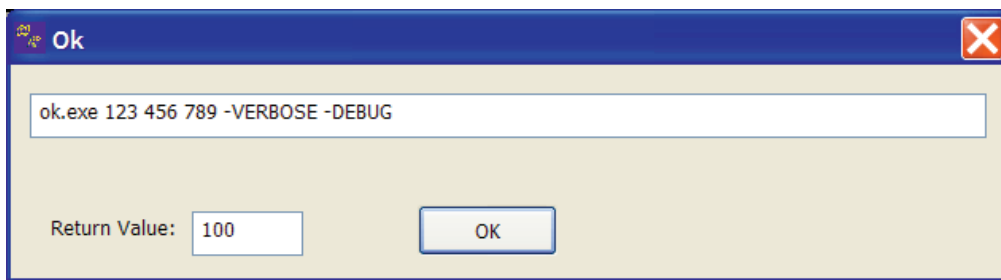


Figure 6-23. Display of the OK program

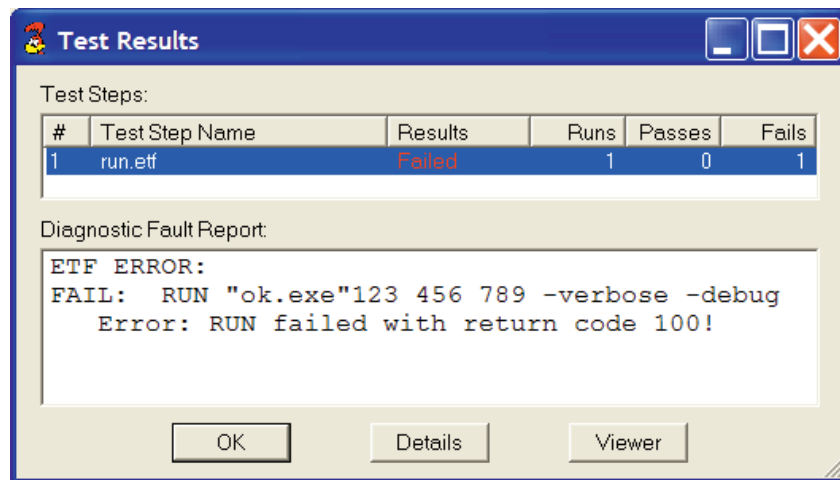


Figure 6-24. RUN command Test Results Dialog - FAIL

Figure 6-24 shows what happens when the external command returns a non-zero status code. In the example above the OK program returns a value of 100. The RUN command determines that the result code is not zero and issues a FAIL condition to ScanExpress Runner. The Test Results dialog shows the failed RUN command, as well as the return code value.

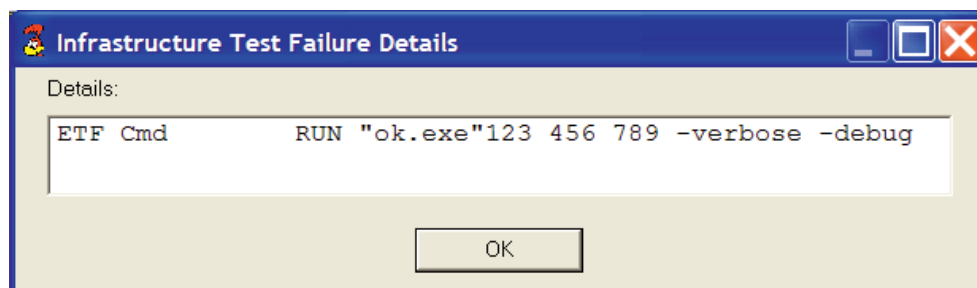


Figure 6-25. Details dialog after OK returns a zero.

Figure 6-25 displays the Details dialog after the OK program returns zero for the status return code. Even though the Test Results dialog has no information for a successfully PASSED test step, the Details dialog will still contain each ETF command processed.

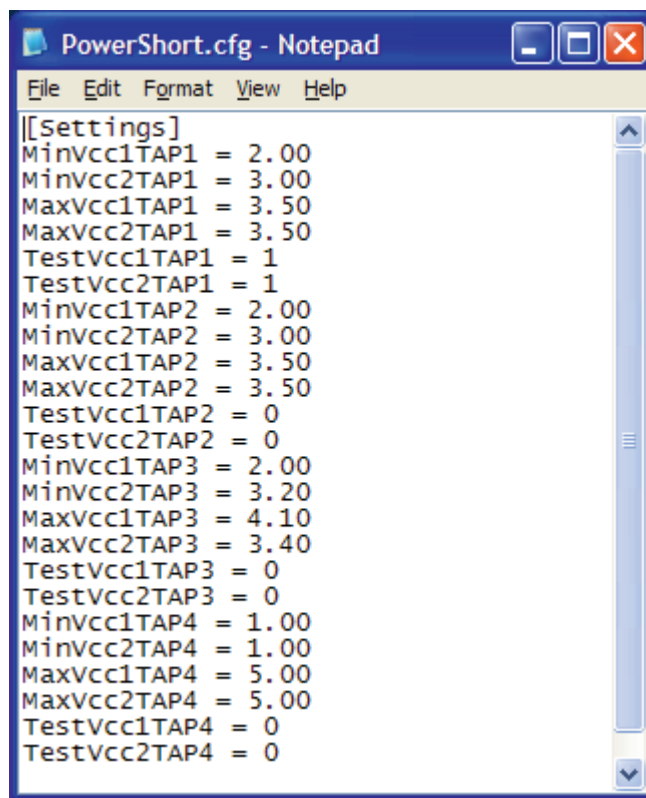
POWERSHORT Command

A test plan may include a Power Short test as the first test step to check that the VCC signal is not shorted to ground. The PowerShort test is implemented using an ETF script file containing a single POWERSHORT command.

The POWERSHORT command executes a PowerShort test for each UUT that checks that the VCC signal is not shorted to ground. A *Test Settings Configuration File* identifies the TAP VCC signals that will be tested and specifies the valid VccMin and VccMax values to be used for the test. The PowerShort test consists of two parts, the Power Off test and the Power On test. The Power Off test, conducted with the UUT power turned Off, checks that the VCC signal is not shorted to ground. The Power On test, conducted with the UUT Power On, performs a connection test to verify that each UUT is connected. The connection test will be performed for each enabled TAP voltage signal as indicated by the *Test Settings Configuration File*, using the specified VccMin and VccMax values. If the voltage level is not measured within the specified range, the test will return a “Failed” result.

The *Test Settings Configuration File* can be created in standard ASCII text format using a program like Notepad or it can be automatically generated from the ScanPlus PowerShort utility.

Figure 6-26 shows an example *Test Settings Configuration File* with Vcc1 and Vcc2 on TAP1 enabled for PowerShort testing. Note that a configuration file created with the ScanPlus PowerShort utility will include a [Controller] section at the beginning of the file. ScanExpress Runner ignores the [Controller] section if it is there and only uses the [Settings] section for the PowerShort test.



```
[[Settings]
MinVcc1TAP1 = 2.00
MinVcc2TAP1 = 3.00
MaxVcc1TAP1 = 3.50
MaxVcc2TAP1 = 3.50
TestVcc1TAP1 = 1
TestVcc2TAP1 = 1
MinVcc1TAP2 = 2.00
MinVcc2TAP2 = 3.00
MaxVcc1TAP2 = 3.50
MaxVcc2TAP2 = 3.50
TestVcc1TAP2 = 0
TestVcc2TAP2 = 0
MinVcc1TAP3 = 2.00
MinVcc2TAP3 = 3.20
MaxVcc1TAP3 = 4.10
MaxVcc2TAP3 = 3.40
TestVcc1TAP3 = 0
TestVcc2TAP3 = 0
MinVcc1TAP4 = 1.00
MinVcc2TAP4 = 1.00
MaxVcc1TAP4 = 5.00
MaxVcc2TAP4 = 5.00
TestVcc1TAP4 = 0
TestVcc2TAP4 = 0
```

Figure 6-26. Test Settings Configuration File

The Power Off Test checks that the VCC signal is not shorted to ground. In order for this test to take place, the user must make sure that the UUT's power is off. The application will prompt the user to turn power OFF on each UUT as shown in Figure 6-27.

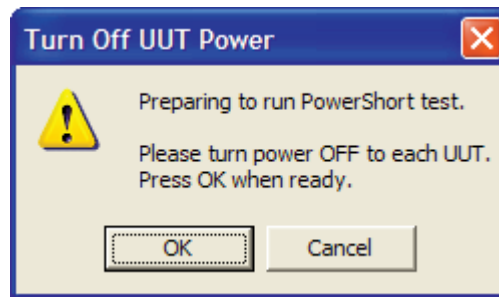


Figure 6-27. PowerOff Test Popup Dialog

A PowerShort test will be performed for each enabled TAP voltage signal as indicated by the **Test Settings Configuration File**. If PowerShort detects that VCC power is on, then the test for that VCC signal will be aborted. The user will be prompted to shut off corresponding UUT's power and restart the test.

The Power On Test verifies the connection to the UUT and requires each UUT to be powered up. Since the Power Short Test requires that the UUT's power to be off, if the UUT is disconnected, then the Power Short Test will pass. The Connection Test is needed to make sure that the UUT is indeed connected. In this step, the user will be prompted to turn on the power to all UUTs as shown in Figure 6-28

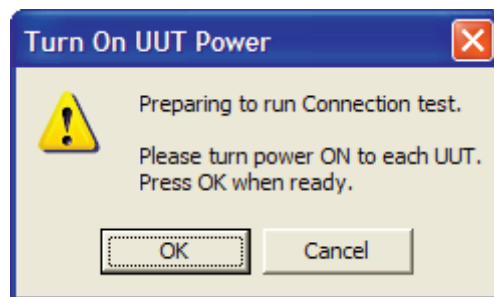


Figure 6-28. PowerOn Test Popup Dialog

A connection test will be performed for each enabled TAP voltage signal as indicated by the **Test Settings Configuration File**, using the specified VccMin, VccMax values from the **Test Settings Configuration file**. If the voltage level is not measured within the specified range, the test returns a "Failed" result.

Test results for the PowerShort test are displayed in the Diagnostic Fault Report pane of the Runner Test Results screen as shown in Figure 6-29. Item 1 indicates the overall PowerShort test result. Item 2 indicates the ETF PowerShort command result. Item 3 indicates the result of the PowerOff test, and is followed by the PowerShort test result of each TAP Vcc signal. If the PowerOff test

passes, the PowerOn test is performed next. Item 4 indicates the result of the PowerOn test, and is followed by the Connection test result and the associated measured Vcc1 and Vcc2 voltage levels for each TAP.

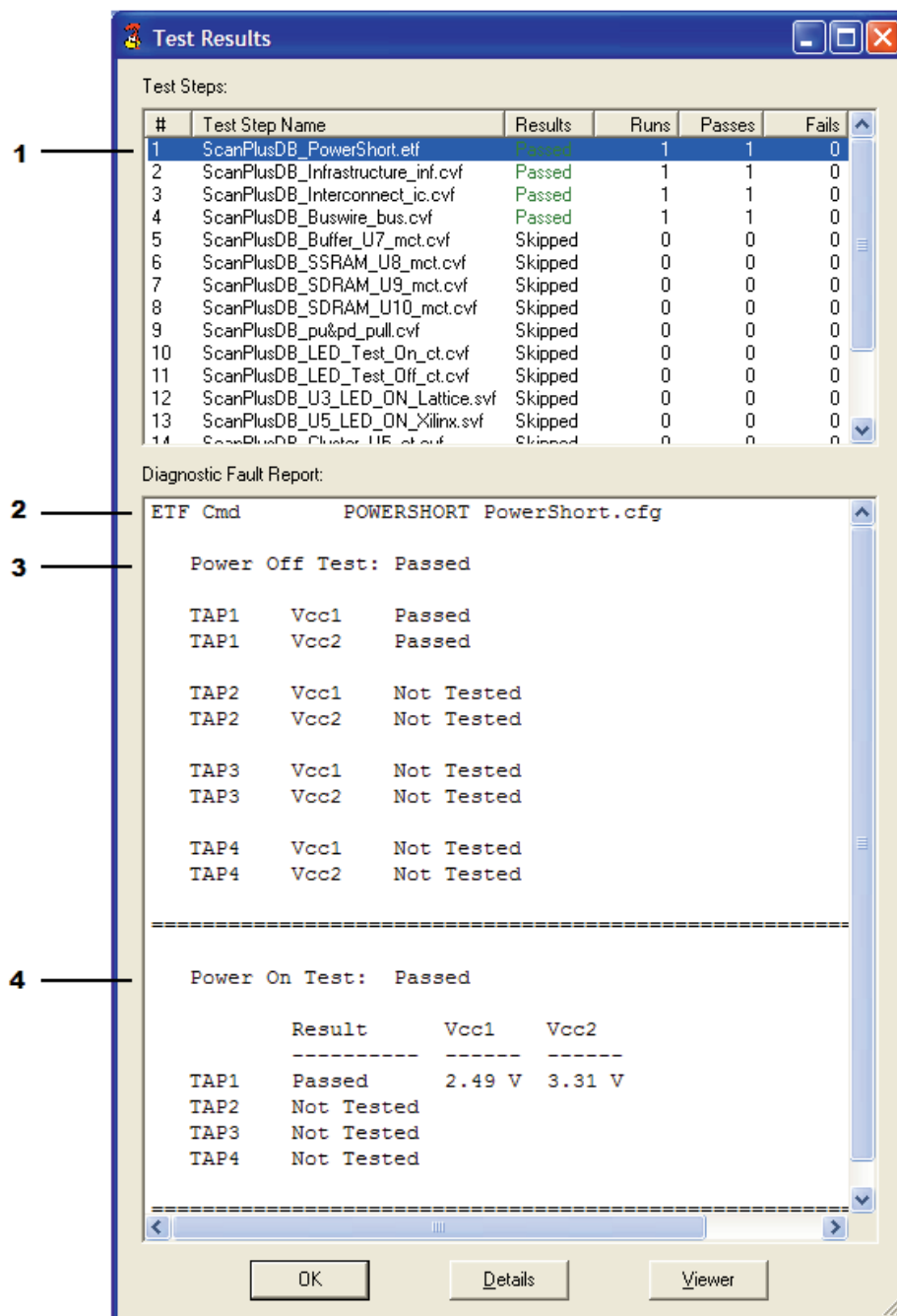


Figure 6-29. Results of the POWERSHORT command

For additional information on the PowerShort test, refer to the ScanPlus PowerShort User's Manual.

Command Reference:

#

This is the comment command. Anything after the '#' up to the end of the current line is considered a comment and ignored by the ETF parser.

Usage:

{Command {parameter {, parameter} ... } ; } # *comment*

Example:

This is an example of a full line comment

SET PIO; # Tri-state the parallel outputs.

CVF

This command will load and execute a CVF file. The CVF file is treated as if it were entered directly into a test plan. A Failure condition will cause the ETF test step to report a failure. If the appropriate options are installed then Detailed Diagnostic information is available upon failure.

Usage:

CVF *filename*;

Return Value:

Pass or Fail.

Parameters:

filename

A fully qualified path and file name of the CVF file.

Example:

CVF C:\Boards\NewProduct\Product101_inf.cvf;

RUN

This command will load and execute an external Windows or DOS program. ETF file parsing and execution will halt until the external program has completed. The external program may return an error status in the DOS ERRORLEVEL. If the return status is non-zero, the ETF file will be treated as a “Fail” condition. A zero return status indicates a “Pass” condition.

Usage:

RUN *command*

Return Value:

Pass or Fail.

Parameters:

command

The name and parameters for an external Windows or DOS program.

Example:

RUN C:\Boards\NewProduct\externaltest.exe /t /b /delay=100; # Run my test program

SET PIO

This command is used to control the state of the parallel input and output pins on the PCI 1149.1 controller. The PIO is split into two eight-bit ports, labeled A and B. The pins in each port are identified as 0 through 7. Thus the available pins are A0, A1, A2,..A7 and B0, B1, B2,..B7.

If not tri-stating the default condition is to drive all unspecified pins to logic “high”. To set a pin to logic “low” precede the pin identifier with an exclamation point. (i.e. !A2)

SET PIO with no parameters will tri-state the output pins.

Usage:

SET PIO { *output*, { *output*, ...} } ;

Return Value:

None

Parameters:

output

The port and pin designator. Valid designators are A0, A1, A2,..A7 and B0, B1, B2,..B7. A designator may be preceded with a ‘!’ to indicate a logic “low”.

Example:

```
set pio;           # Tri-State the PIO (Note the command is case insensitive)
SET PIO a1;        # Drive ALL the outputs high. ('1' is the default condition for all bits)
set pio a0,!b0,     # Drive pin zero of port A to logic “high”, bit 0 port B to “low”
    !a1, b1,        # Note that there is no semi-colon yet. That means the command
    a2,!b2 ;        # can extend to multiple lines, and you can comment each line
```

The final result will be:

Port A	PortB
MSB...LSB	MSB...LSB
11111101	11111010

SET DMM

This command is used to set the mode for the digital multi-meter. The mode consists of the function, range, and rate. Normally you will not have to set a rate value. This is primarily used to set a rate for programmatic multiple readings of the DMM, and is the rate in samples per second. The function determines what type of measurements the DMM will make. The valid types of measurements are OHMS, VOLTSAC, VOLTSDC, CURRENTAC, and CURRENTDC. The range selects the measurement range of the DMM. The range is a numeric value and may include units and/or decimal modifiers. Units (such as V for volts, or A for amps) are ignored, and used only for clarity in expression. Decimal modifiers are applied to the numeric value to provide a “normalized” number. For example, 10K is equivalent to 10000.0, and 250mV is equivalent to 0.25.

Usage:

SET DMM *function* **RANGE**=*range* { **RATE**=*rate* };

Return Value:

Pass, or Fail if unable to initialize the DMM.

Parameters:

function

One of the following: OHMS, VOLTSAC, VOLTSDC, CURRENTAC, CURRENTDC.

range

The range varies with the function (Volts, Ohms, or Amps). Valid ranges are:

Volts: 300mV, 3V, 30V, 300V

Amps: 3mA, 30mA, 300mA, 3A

Ohms: 30, 300, 3K, 30K, 300K, 3Meg, 30Meg, 300Meg

Note: If you use a decimal modifier (such as mA, or Meg) there must be **no space** between the numeric value and the modifier. Thus 30Meg is correct, and 30 Meg is not.

rate

Numeric value indicating the number of readings per second allowed. Minimum value is 1, maximum is 1000. The default (and recommended value) is 10.

Example:

Note that the commands are all case-insensitive.

SET DMM CURRENTDC RANGE=300mA ; # Could have said .3 instead of 300mA

Set dmm voltsdc range=30v;

set dmm ohms range=300000 ; # Same as saying 300K

SET RELAY

This command controls the relays on the two 40 channel relay PCI cards. To turn on a relay, select the card number and relay number. Combine multiple relays with a single SET command. Any relay not specifically designated in a SET RELAY command will be released to the open state. A SET RELAY command with no parameters will release all of the relays.

Usage:

SET RELAY { *card*, *relay* { ,*card*,*relay*... } };

Return Value:

Pass, or Fail if unable to initialize the DMM.

Parameters:

card

Relay controller card. 1=first PCI relay card, 2=2nd PCI relay card

relay

Relay identifier. 1-40 for Channel relays, or name of configuration relay:

Configuration Relay Names:

AtoA, BtoA, BtoB, CtoA, CtoC, DtoA, DtoC, DtoD

Examples:

```
set relay;                # Releases all relays on both cards
set relay 1,1,2,1;        # Turn on relay 1 on card 1, relay 1 on card 2

set relay 1,1,
    2,1 ;                 # Same as previous, just easier to read

set relay 1,AtoA,         # It is possible to have comments on each line
    1,CtoA,              #
    1,1,                  #
    1,12,                 #
    1,32,                 #
    2,AtoA,               #
    2,39,                 # Splitting a complex command over several lines
    2,40;                 # makes it more readable to the human.
                        # The parser doesn't care!
```


READ DMM

This command reads the Digital Multi Meter and optionally saves the value into a variable. It can also optionally perform a pass/fail comparison against an expected value and a tolerance, or against a Min and Max value.

Usage:

```
{var = }READ DMM {expected, tolerance} {MAX=xxx, MIN=xxx};
```

Return Value:

Pass, or Fail if comparison feature is active and the value is not within the specified limits.

Parameters:

var variable in which to save the numeric value

expected
 expected value (numeric)

tolerance
 Tolerance to compare against "+/- value". A tolerance may be a simple numeric, or it may use decimal modifiers. Appending the per-cent sign will cause the tolerance to be calculated as a per-cent of the expected value.

MIN=xxx
 Minimum acceptable value for the test to pass.

MAX=xxx
 Maximum acceptable value for the test to pass.

Examples:

```
READ DMM 12.0V +/- 500mV;     # Passing values would be 11.5V to 12.5V
```

```
# if in Resistance mode:
```

```
READ DMM 32.5K +/- 10%;     # 10 percent tolerances against a 32.5k ohm resistor
```

```
# or current mode
```

```
READ DMM MIN=10mA, MAX=250mA;   # must be between 10 and 250 mAs to pass
```

READ PIO

The read PIO command reads the input pins of the PCI 1149.1 controller, and stores them in a variable. The value is stored as a 16 bit value with PORT B in the upper 8 bits.

An optional MASK can be specified to strip undesired bits from the result. Only the pins corresponding to a '1' in the mask will be included in the result. The mask is a 16 bit value, with bit zero of the mask corresponding to bit zero of the returned value. The mask is used primarily to allow numerical comparisons of the state of a subset of the PIO pins.

Usage:

```
{Var = }READ PIO {MASK=nnnn};
```

Return Value:

Pass, or Fail if PIO is unable to be read.

Parameters:

Var

Variable in which to save the numeric value

MASK=expected

Bit mask to determine which input pins will be used in the result. A '0' bit in the mask causes the corresponding PIO bit to be ignored.

tolerance

Example:

```
A = READ PIO MASK=0x7F;      # Only 11 bits are significant, bits 0-10.  
                              # This allows us to ignore non-connected pins.
```

WAIT PIO

The wait command polls the PIO input bits waiting until they match the desired state. To build a state mask you will specify each pin that you wish to consider, and the desired polarity. Only the pins that you specify are significant in the comparison.

The PIO is split into two eight-bit ports, labeled A and B. The pins in each port are identified as 0 through 7. Thus the available pins are A0, A1, A2,..A7 and B0, B1, B2,..B7. Specifying a pin will cause the WAIT PIO command to wait until that pin has a “high” logic level. Placing a "!" before the pin name will invert the condition, i.e. it will wait for a pin to be “low”. If the desired condition does not occur within TIMEOUT milliseconds the WAIT PIO command issues a “FAIL”.

Usage:

```
{Var = }WAIT inputBit { , inputBit {...} } {TIMEOUT=nnn}
```

Return Value:

Pass, or Fail if PIO is unable to be read, or the timeout value is exceeded.

Parameters:

Var

Variable in which to save the numeric value

TIMEOUT=nnn

nnn is the time out value in milliseconds. If the pins on the PIO do not match the selected pattern within nnn milliseconds, the test will fail.

inputBit

inputBit is the Port and Pin identifier. Acceptable values are: A0-A7 and B0-B7. Placing a "!" in front of the input identifier inverts the condition.

Examples:

```
wait a0,!b1, timeout=500;    # wait up to 500 milliseconds for
                             # PortA0 to go HIGH and PortB1 to go LOW
```

DELAY

The DELAY command delays execution of the next ETF command for a specified number of milliseconds. This is ideal to allow settling time for the circuitry under test.

Usage:

DELAY *nnn*

Return Value:

None.

Parameters:

nnn

nnn is the number of milliseconds to delay.

Example:

DELAY 250; # Pauses for 250 milliseconds

PRINT

This command prints a message in a Windows Message Box. It can be either a message in quotes (a string), or it can be a string variable. Execution of the test step is halted until the user closes the message box.

Usage:

PRINT "string";

PRINT variable\$;

Return Value:

None.

Parameters:

"string"

"string" is a message string enclosed in quotes.

variable\$

variable\$ is a previously defined variable. It may be a string variable or the result of a previous test.

Examples:

```
PRINT "Message";           # This will pop up a Windows Message Box
```

```
Var$="This is a longer message";
```

```
PRINT Var$;
```

```
result$ = READ DMM;
```

```
PRINT result$;
```

COMPARE

Compare subtracts the second input value from the first, and optionally compares the results to a specified value. The input values can be a numeric constant, or a previously defined variable.

The target value can be a numeric value and a tolerance, or it can be defined by setting a minimum and maximum limit. If the result of the subtraction falls within this range the COMPARE command will report a PASS condition. If the result is outside the specified range, the COMPARE command will report a FAIL condition.

Usage:

{*var* =} COMPARE *val1*, *val2*, { *expected*, *tolerance* } or { Min=*xxx*, Max=*xxx* }

Return Value:

Pass, or Fail if comparison feature is active and the value is not within the specified limits.

Parameters:

var

variable in which to save the numeric value of the comparison results.

val1

Value number 1 to compare (numeric OR string variable)

val2

Value number 2 to compare (numeric OR string variable)

expected

expected value (numeric).

tolerance

Tolerance to compare against "+/- value". A tolerance may be a simple numeric, or it may use decimal modifiers. Appending the per-cent sign will cause the tolerance to be calculated as a per-cent of the expected value.

Min=*xxx*

Minimum acceptable value for test to pass.

Max=*xxx*

Maximum acceptable value for test to pass.

NOTE: Use either an expected value and a tolerance, or a Min and Max value for comparison tests. If there is no Min, Max, or expected value the COMPARE command will still perform the subtraction, but it does not report the results as a PASS/FAIL.

Examples:

A = 10.5;

B = 10.6;

C = COMPARE A,B ; # C will contain -0.1

A = Read dmm;

{do some stuff here}

B = Read dmm;

COMPARE a,b, Min=0, Max=.05; # The two values are expected to be
within .05 units of each other

POWERSHORT

This command is used to run a PowerShort test to check that the Vcc signal is not shorted to ground, and a Connection test to check that the UUT is properly connected. If either the PowerShort test fails or the Connection test fails, the POWERSHORT command will report a FAIL condition.

Usage:

POWERSHORT *filename* { *testType* } { *noPrompt* } ;

Return Value:

Pass, or Fail if the PowerShort or the Connect test failed.

Parameters:

filename

The name of the Test Settings Configuration file to be used for the test. The Test Settings Configuration file identifies the TAP VCC signals that will be tested and specifies the valid VccMin and VccMax values to be used for the test

testType

testType identifies the type of power short test to run. If no value is specified for *testType*, both power short tests will be run. Possible values are:

OffTest run the Power Off test – PowerShort test

OnTest run the Power On test – Connection test

noPrompt

Disable display of popup window that prompts the operator to turn off power to the UUTs before running of the PowerOff test, or to turn on power to the UUTs before running of the PowerOn test. This parameter is only valid if the *testType* parameter is also specified.

NoPrompt do not display prompt dialog during test

Examples:

POWERSHORT powershort.cfg; # Run both tests

POWERSHORT powershort.cfg OffTest; # Run the power off PowerShort test

POWERSHORT powershort.cfg OnTest; # Run the power on Connection test

POWERSHORT powershort.cfg OnTest NoPrompt; # Run the test with no prompt

NOTE: A separate ETF file must be used for the power short test. Do not combine the PowerShort ETF command with other ETF commands in the same ETF file.

NOTE: A POWERSHORT command with no *testType* parameter will execute both the PowerOff and PowerOn tests. However the PowerOn test is executed only if the PowerOff test passes.

When the POWERSHORT command is used with a *testType* parameter to execute the PowerOn or PowerOff test separately, it is recommended that separate ETF files be used for each POWERSHORT command. If a command to run the PowerOff test is followed by a command to run the PowerOn test in the same ETF file, and the PowerOff test fails, the PowerOn ETF command will still execute.

MSP430_ENABLE_4WIREJTAG

This command is used to enable 4wire JTAG mode of TI MSP430 devices. Newer generation of MSP430 devices (such as MSP430F2252 or MSP430F5438) can operate in Spy-By-Wire mode or traditional JTAG mode. They power up in Spy-By-Wire mode by default and this command must be executed before programming them using JTAG connection. Currently this command is supported using NetUSB with built-in ScanTAP (NetUSB-1149.1/E and NetUSB-1149.1/SE) controllers only.

Also, pin 11 of the controller must be connected to RST/NMI/SBWTIO signal and pin 13 of the controller must be connected to TEST/SBWTCK signal.

Usage:

MSP430_ENABLE_4WIREJTAG;

Return Value:

None.

Parameters:

None.

Examples:

```
MSP430_ENABLE_4WIREJTAG;           # Enable 4wire JTAG mode
```