

Boundary-Scan Scripting

A member of the ScanExpress family

- ❑ Create your own custom boundary-scan tests and diagnostics
- ❑ Script generation using ScanExpressTPG's new built-in script debugger
- ❑ Scripts are fully compatible with ScanExpress Runner v6.0
- ❑ Customize test messages to the screen and/or to a log file
- ❑ Script debugger supports high level script development features:
 - ❑ Execute and debug script while editing
 - ❑ Color-coding
 - ❑ Breakpoints
 - ❑ Single step
 - ❑ Variable watch window
- ❑ Powerful C style syntax is easy to use and supports branching and looping
- ❑ Script syntax includes dedicated scan functions:
 - ❑ Group pins
 - ❑ Scan pin values
 - ❑ Set pin values
- ❑ Typical applications:
 - ❑ Program custom protocol EEPROM
 - ❑ Write custom data to a log file
 - ❑ Test DAC to ADC loopback and check for a range of values
 - ❑ Custom instructions to operator based on test results

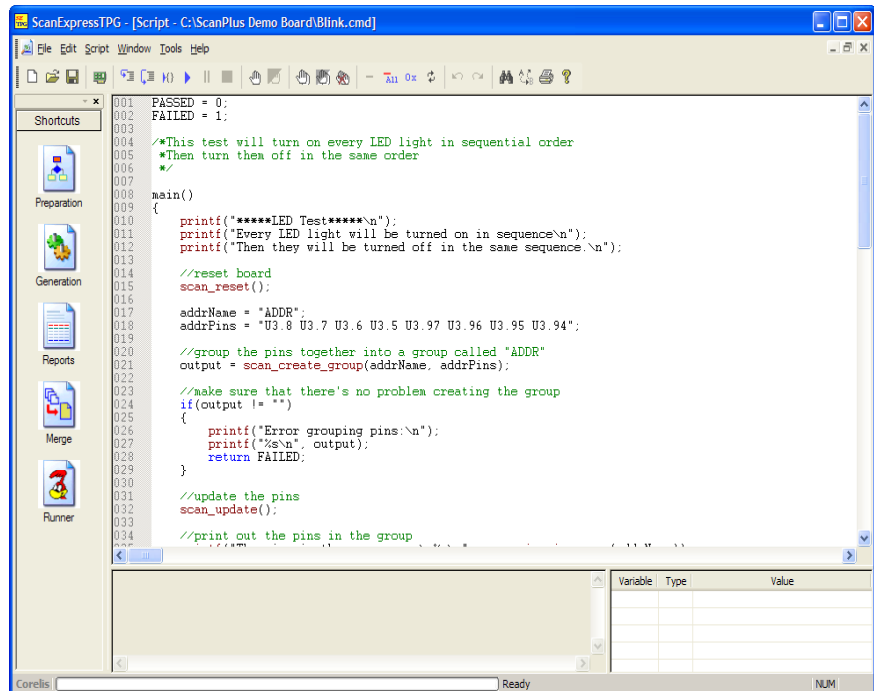


Figure 1. Script Debugger Window in ScanExpressTPG

Introduction

Boundary-scan Scripting is a new feature that enables the user to create customized boundary-scan testing sessions. ScanExpress Runner DLL, Runner ETF files, ScanExpress Debugger DLL and cluster testing are now joined by a powerful script engine that supports even higher levels of test customization.

The new ScanExpressTPG and ScanExpress Runner tools now support integrated boundary-scan scripting capabilities which are available to customers with a valid support agreement at no additional charge.

ScanExpressTPG now incorporates a new GUI for creating and debugging script tests for use in ScanExpress Runner.

For example, boundary-scan scripting can be used to perform custom tests such as programming a spe-

cial EEPROM device using a proprietary serial protocol, checking ADC readings within a valid range of values, testing custom memory not covered by a standard memory test, and saving user customized test data and messages to a log file.

The script command files are user-generated text files that perform various scan-related functions. The syntax is very much like 'C' and is similar to the style and format of other Corelis products that support scripting. The script commands also contain ScanExpress Debugger-like scan commands. The built-in Script Debugger assists in script manipulation by providing syntax highlighting and syntax checking.

The boundary-scan scripting functionality is integrated into both ScanExpressTPG and ScanExpress Runner. The files required to create scripting test steps are the same as other test step types in ScanExpressTPG. The files generated by

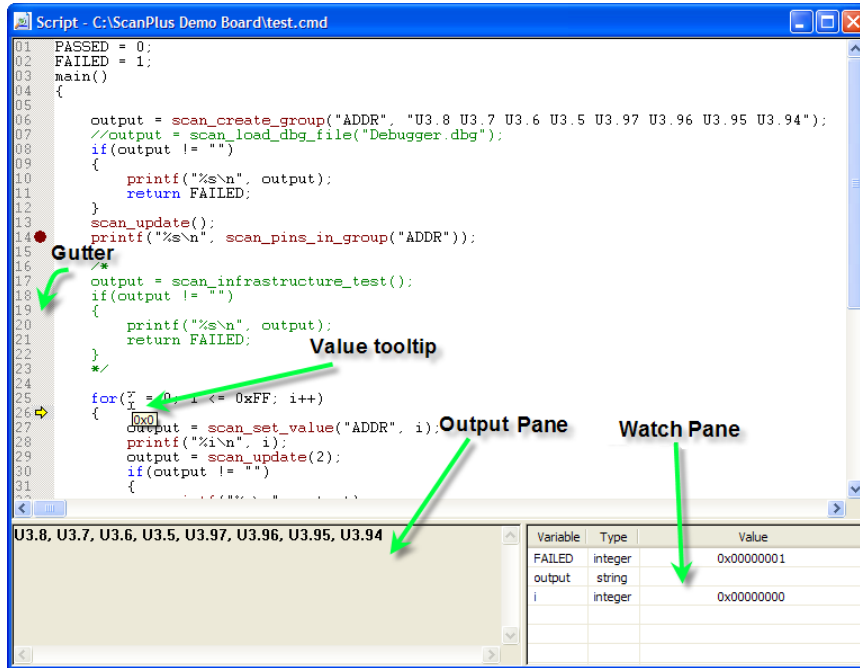


Figure 2. Script Debugger Window

ScanExpressTPG are also the same as other test step types and can be directly used as test steps in ScanExpress Runner.

A typical script command file consists of sequential scanning instructions which performs reads and writes, delays, setting parallel I/O signals, looping, conditional branching, printing to the screen and printing to a file. The script is also able to provide a PASS/FAIL indication.

Users can manually debug a script in ScanExpressTPG by selecting the specific test step containing the script and clicking on the “Debug” button which then invokes the Script Debugger window. The user may use the Script Debugger to edit the script, single step, set breakpoints, run, watch variables and query values.

Features and Uses

Boundary-scan scripting complements other boundary-scan solutions from Corelis. With the powerful scripting capability, the user can easily create tests that previously required writing customized code before using the Scan Function Library. Pin-level control combined with a scripting language now allows the user flexibility while keeping programs short and understandable.

New uses for boundary-scan scripting include:

- Programming devices and peripherals using custom protocols not yet supported by other Corelis tools.
- Testing devices such as ADC where the value is not an exact value but falls within a range.
- Setting pin values based on values of other pins.
- Recording custom values into log files.
- Customizing diagnostic messages.

Script Development

ScanExpressTPG version 1.14 and later provides integrated support for scripting. Users can use ScanExpressTPG to edit and generate test scripts, which can then be added directly in ScanExpress Runner as standard test step.

In the script creation process, the standard ScanExpressTPG topology, netlist, and constraint files are automatically associated with the script, allowing the scripting environment to understand the boundary-scan chain information.

Script Debugger

When the user debugs a script test, a window similar to Figure 2 is opened. The Script Debugger is a text editor that allows the user to develop and test the script before generating the test for use in ScanExpress Runner. The Script Debugger has the following capabilities to facilitate script development and testing:

- No compilation needed to execute script
- Color coding of built-in functions and keywords
- Starting, stopping, and pausing program execution
- Setting breakpoints
- Single-stepping into or over function calls
- Stepping into or over an arbitrary number of lines
- Displaying variable values while the script is executing.

The Script Debugger contains an output window to display messages generated during script execution. The watch window allows the user to add variables for a list to watch as the script progresses. The user can also view the value of any variable while the script is paused by hovering the mouse over its name. The gutter on the left of the window indicates the script status such as the currently executing line number and which lines have breakpoints.

With the Script Debugger, the user can develop proper code even when the target device is not present. Keyword color-coding shows the user in real time whether certain words and function calls are recognized. As with any text editor, tools for finding and replacing text as well as cut, copy, paste, and print are included.

Scripting Language

The scripting language is a simplified language modeled after C. A person familiar with C can easily write scripts without a steep learning curve.

The interpreted scripting language contains support for looping (`for` and `while` loops), conditional branching, and many other features one would expect in a language

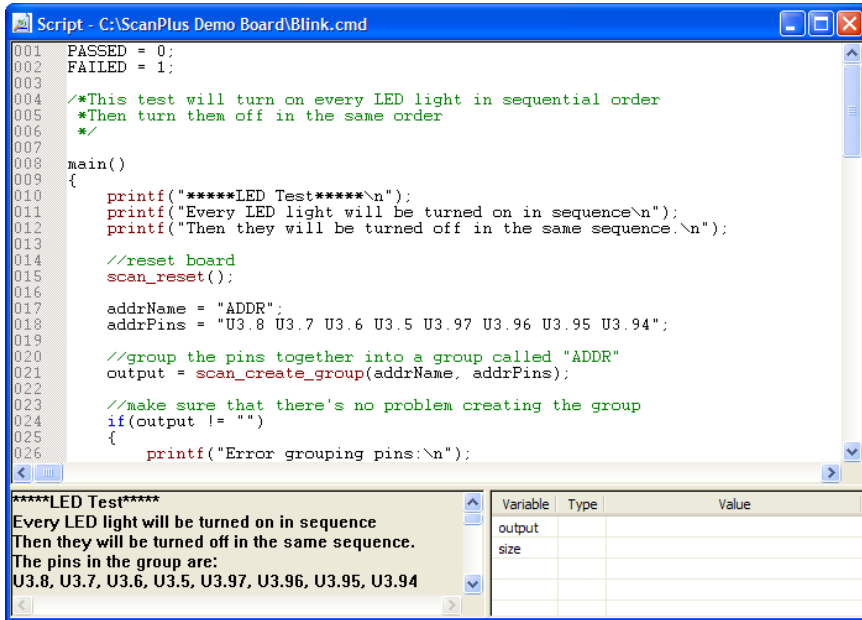


Figure 3. ScanExpressTPG Test result

similar to C. Because the variables are dynamically type-cast at run-time, it is unnecessary to declare variable types.

The scripting language contains built-in functions that allow writing text to a file, outputting messages,

and pausing execution for a predetermined period. Boundary-scan-specific functions allow the user to group pins, read pin values, and set pin values. Most of the functionality supported by ScanExpress Debugger is available in the boundary-scan scripting.

Each script command file contains one or many functions. The user can create subroutines and call them from the main function, thus enabling repetition of commonly-used tasks.

Table 1 on the next page provides a listing of built-in functions supported by boundary-scan scripting.

Script Execution

Both ScanExpressTPG and ScanExpress Runner allow the execution of boundary-scan script test steps. When the script is executed from within ScanExpressTPG, the output window displays the output messages specified by the script, as shown in Figure 3. Users can pause the script, query variable values, and set breakpoints. The user can also step through every line of the script to determine where an error occurs.

Controller Support

ScanExpressTPG Script Debugger and ScanExpress Runner work with all of the boundary-scan controllers currently supported by Corelis. Us-

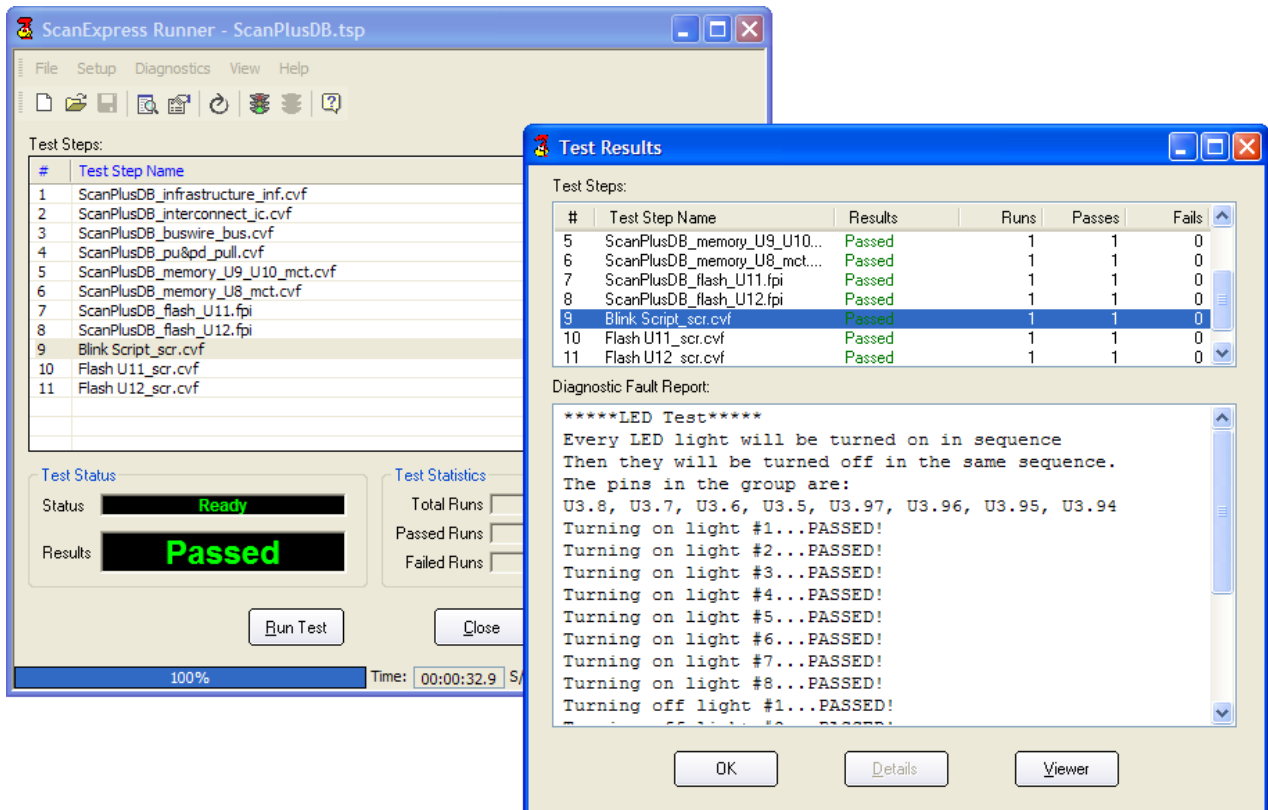


Figure 4. ScanExpress Runner Test result

Function	Description
<code>delay()</code>	Pauses execution of the script for a specified number of milliseconds.
<code>fclose()</code>	Closes a currently open file.
<code>fcloseall()</code>	Closes all currently open file.
<code>fgetline()</code>	Reads a line from an open file and formats it.
<code>fopen()</code>	Opens a file for reading or writing.
<code>fprintf()</code>	Writes a formatted string to a file.
<code>print()</code>	Outputs a string to the output buffer.
<code>printf()</code>	Outputs a formatted string to the output buffer.
<code>progress()</code>	Updates the state of the progress of the script.
<code>random_integer()</code>	Generates and returns a pseudorandom number.
<code>scan_check_conflict()</code>	Check the user input value for conflicts
<code>scan_create_group()</code>	Creates a group of pins
<code>scan_delete_group()</code>	Deletes the group specified
<code>scan_get_value()</code>	Returns the value sensed by the input pin (or group)
<code>scan_infrastructure_test()</code>	Performs the infrastructure test on the UUT.
<code>scan_load_dbg_file()</code>	Loads a .dbg file generated by the ScanExpress Debugger.
<code>scan_load_files()</code>	Loads the required topology file as well as optional netlist file and fixed outputs file if specified.
<code>scan_num_pins_in_group()</code>	Returns the number of pins in the specified group.
<code>scan_pins_in_group()</code>	Returns the list of pins.
<code>scan_set_value()</code>	Assigns a value to the output pin or group.
<code>scan_reset()</code>	Resets the hardware.
<code>scan_update()</code>	Updates the scan chain with user set values and retrieves the new input values.
<code>seed_random()</code>	Sets a starting point for the pseudorandom number generator.
<code>string_compare()</code>	Returns an integer representing the lexicographical relation between two strings.
<code>string_concatenate()</code>	Returns the concatenation of two strings.
<code>string_format()</code>	Takes a formatting string and a variable number of arguments as inputs and returns a formatted string.
<code>string_substring()</code>	Finds the first occurrence of a specified substring from within a string and returns the string starting from that point.
<code>string_to_float()</code>	Converts a string decimal representation of a floating point value into a floating point output.
<code>string_to_integer()</code>	Converts a string decimal representation of an integer into an integer output.

Table 1. Built-in scripting functions

ers can change the controller type directly from the tool GUI without having to edit the script. This allows greater flexibility for the user.

ScanExpress Runner Integration

Once script development is completed, users can compile the script test steps from within ScanExpressTPG. The resulting generated test step can then be directly loaded into ScanExpress Runner for execution.

Compiled scripts are executed like other tests in ScanExpress Runner. When the script is executed, it will indicate whether the test has passed, using information provided within the script. Users can easily view the diagnostics message by double-clicking on the test step. An example diagnostic message window is shown in Figure 4.

Example Scripts

ScanExpressTPG includes several script examples to demonstrate powerful scripting capabilities. These examples perform tests on the ScanPlus Demo Board, an optional test target board supplied by Corelis. Script examples include testing LEDs, flash and SSRAM memories.

As the script examples in Figures 5 and 6 show, boundary-scan test scripts can be portable and easily understood without sacrificing flexibility. If an underlying protocol is changed, only the functions implementing that protocol need to be changed instead of the whole program.

LED Blink

The first example, whose source is partially shown on Figure 5, toggles the LED lights on the ScanPlus Demo Board. It will report errors if detected.

U8 SSRAM Memory Test

This script example writes to the SSRAM memory of the ScanPlus Demo Board. It writes a walking 1 pattern to certain address locations and reads them back to make sure that the memory is correctly written. The source code for this example is partially shown in Figure 6.

U11 Flash Test

This script example programs and tests an AMD Flash device. It first

```
main()
{
    printf("*****LED Test*****\n");
    printf("Every LED light will be turned on in sequence\n");
    printf("Then they will be turned off in the same sequence.\n");

    //reset board
    scan_reset();

    addrName = "ADDR";
    addrPins = "U3.8 U3.7 U3.6 U3.5 U3.97 U3.96 U3.95 U3.94";

    //group the pins together into a group called "ADDR"
    output = scan_create_group(addrName, addrPins);

    //make sure that there's no problem creating the group
    if(output != "")
    {
        printf("Error grouping pins:\n");
        printf("%s\n", output);
        return FAILED;
    }

    //update the pins
    scan_update();

    //print out the pins in the group
    printf("The pins in the group are:\n%s\n", scan_pins_in_group(addrName));

    //flash every single LED
    size = scan_num_pins_in_group(addrName);
    ON = 1;
    mask = 0;

    //turning on LED in order
    for(i = 1; i <= size; i++)
    {
        //set the mask to turn on the LED indicated by i
        mask = (mask << 1) | ON;

        printf("Turning on light #%i...\n", i);
        if(set_and_check(addrName, mask) == FAILED)
        {
            printf("FAILED!\n");
            return FAILED;
        }
        else
            printf("PASSED!\n");

        //sleep for 100 ms
        delay(100);
    }

    //Turning off the light in order
    for(i = 1; i <= size; i++)
    {
        //get rid of the least significant i bits
        mask = (mask >> i) << i;

        printf("Turning off light #%i...\n", i);
        if(set_and_check(addrName, mask) == FAILED)
            return FAILED;
        else
            printf("PASSED!\n");

        //sleep for 100 ms
        delay(100);
    }

    printf("All tests PASSED!\n");
    return PASSED;
}
```

Figure 5. LED Blink script source

performs an ID check, then erases the chip before performing a blank check. Next, it programs several locations with a certain value and reads it back. If the value it read does not match the value it wrote, it reports an error.

U12 Flash Test

This script example programs and tests an Intel Flash device. It performs the same tasks as the U11

Flash Test, and shares the same main function, but the auxiliary functions that implement the Flash programming protocol are changed to reflect the Intel protocol.

```

main()
{
    printf("*****U8 MEMORY TEST*****\n");

    if(setup())
        return FAILED;
    for(i = 0; i < 15; i++)
    {
        value = 1 << i;
        printf("Clearing location 0x%04x with 0xFFFF...", value);

        // Write the location value with FFFF
        if(write(value, 0xFFFF))
            return FAILED;
        // Make sure that data received is FFFF
        received = read(value);
        if(received != 0xFFFF)
        {
            printf("FAILED!\n");
            printf("\tExpected: 0xFFFF, Received: 0x%04x\n", received);
            return FAILED;
        }
        printf("PASSED!\n");
        printf("Writing 0x%04x to location 0x%04x\n", value, value);
        // Write the location value with value where value is a walking 1 pattern
        // (i.e. 0x1, 0x2, 0x4, etc.)
        if(write(value, value))
            return FAILED;
    }
    for(i = 0; i < 15; i++)
    {
        value = 1 << i;
        printf("Reading memory location 0x%04x...", value);
        // Verify that the location value contains value where value is a walking 1 pattern
        // (i.e. 0x1, 0x2, 0x4, etc.)
        received = read(value);
        if(received != value)
        {
            printf("\nError reading memory location 0x%04x:\n", value);
            printf("\tExpected: 0x%04x, Received: 0x%04x\n", value, received);
            return FAILED;
        }
        else
        {
            printf("PASSED!\n");
        }
    }

    postamble();
    printf("PASSED!");
    return PASSED;
}

```

Figure 6. SSRAM Memory Test script source

CORELIS

13100 Alondra Blvd.
Cerritos, California 90703
 Tel: (562) 926-6727, Fax: (562) 404-6196
sales@corelis.com www.corelis.com

Windows 2000/XP are trademarks of Microsoft Inc.

CodeRunner, DirectWrite, ScanPlus Flash, ScanPlus, ScanExpress, ScanExpressTPG, ScanExpress Runner, USB-1149.1/E, and NetUSB-1149.1 are trademarks of Corelis Inc.

All other trademarks referred to herein are the property of their respective owner.

© Copyright 2007-2010 by Corelis Inc. All rights reserved.

CORELIS Inc., reserves the right to make changes in design or specification at any time without notice.