

WEB/PHP BASED DISTRIBUTED BUILT-IN SELF-TEST MANAGEMENT

Enyedi Sz., L. Miclea

*Technical University of Cluj-Napoca, Automation Department
Barițiu str. 26-28, Cluj-Napoca, Romania
Szilard.Enyedi@aut.utcluj.ro, Liviu.Miclea@aut.utcluj.ro*

Abstract

This paper presents a concept and experimental results on WEB/PHP based distributed built-in self-test (DBIST) management. The devices and tests are selected in an electronic web form, using an ordinary web browser, which then sends the selections to the DBIST server. The server uses PHP scripts to start the selected tests on the selected devices. The scripts use TCP/IP to communicate with the BIST modules of the devices. The BIST modules run the required tests and return the results to the scripts, which generate a web page with the test results and send it back to the requesting client (i.e. the user's web browser).

In our experimental implementation, the PHP scripts generate a human-readable "test results" page, but in future implementations, they can use raw TCP/IP or other protocols to communicate with a central administrative authority or other DBIST networks, interact with SQL databases or do other high or low level test configuration/test results management.

Keywords

BIST, DBIST, WEB, PHP, TCP/IP

1. INTRODUCTION

1.1. Generalities

This paper continues the work [1] of the authors for developing solutions for DBIST systems. However, instead of CORBA or DCOM, we use PHP to give the user

possibility for test selection, communicate with the BIST modules, collect the tests' results and display them for the user.

The example discussed here is a basic one, but allows a good understanding of the principles.

1.2. DBIST

Built-in self-test has been around for quite a while. One of the current trends in BIST technology is Distributed BIST, or DBIST. The distributed nature of DBIST means that each of the modules in the DUT has its own BIST routine, which runs the test more or less independently from the other modules. This way, the actual BIST of the whole device is decomposed into smaller, dedicated BISTs, which should be simpler and easier to develop and maintain.

1.3. PHP

1.3.1. Generalities

PHP (Personal Homepage Preprocessor) is a scripting language, used mostly on web servers. The idea is that the PHP commands are embedded in the HTML file itself. When a browser requests a page from the web server, the server first runs the PHP interpreter on the requested file, and then sends the results to the requesting browser.

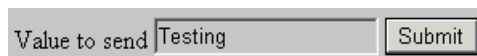
1.3.2. Example 1

```
<html>
<body>
<?php
echo "Test text.<br>" ;
?>
</body>
</html>
```

When the above example is parsed by the server's PHP engine, the `<?php ... ?>` tag is replaced by the result of the contained PHP commands, i.e. by the string "Test text.
". Only then is the resulted page sent to the browser that requested it.

1.3.3. Example 2

This example displays a form in a web page, as shown in figure 1.



The image shows a web form with a text input field and a submit button. The text input field is labeled "Value to send" and contains the text "Testing". The submit button is labeled "Submit".

Figure 1 – the form as displayed in the browser; the user just typed in "Testing".

And its source:

```
<form action="answer.php" method="post">
Value to send <input type="text" name="val">
<input type="submit">
</form>
```

When the user clicks on the “Submit” button, the form data is sent to the server, which passes it to the “answer.php” script, shown below:

```
<html>
<head>
  <title>Answer page</title>
</head>
<body>
  The received value is: <?php echo $val; ?>.
</body>
</html>
```

The “answer.php” script receives the variables of the form and their filled-in values. The script outputs a web page, in which the PHP tag has been replaced by the result of the PHP code, that is the value of the received variable *val*, which is in fact the text the user typed into the text box of the form. This “on the fly” webpage is then sent back to the user, who will see in the browser window the text “The received value is: Testing”.

1.3.4. Combining DBIST with PHP

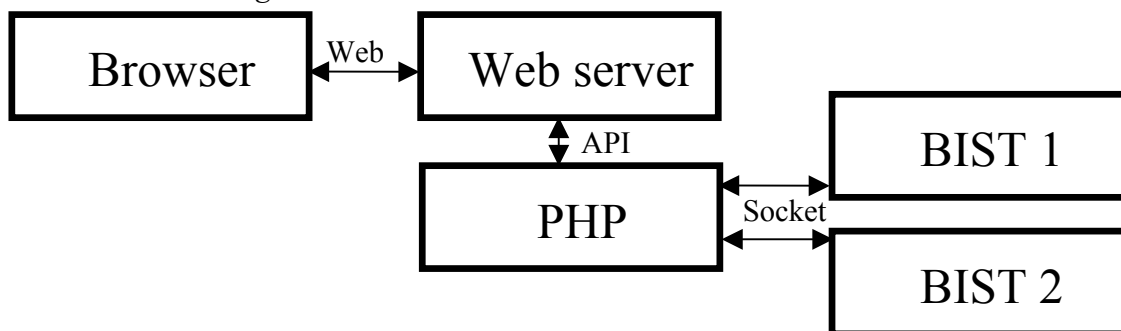


Figure 2 – The structure of the DBIST system using PHP

The steps of the DBIST process:

- The browser requests from the Web server the page containing the PHP script;
- The server (actually, the PHP script) creates on the fly the form and sends it to the requesting browser;
- The user fills the data in the form and sends it back to the server (by clicking on the “Submit” button);
- The PHP script on the server opens a TCP/IP socket, connects to the specified BIST module of the specified device and sends it the user-requested test;
- The contacted BIST module runs the requested test and returns the results to the script, through the same TCP/IP connection;
- The script generates the results page and sends it to the user’s browser, which displays it.

2. IMPLEMENTATION AND EXPERIMENTS

2.1. *The experimental BIST module*

The simplest BIST module [1] that we used in the experiments was a software module, written in Visual Basic. The tests it can run check the available space on the disk drives, their share names and other details.

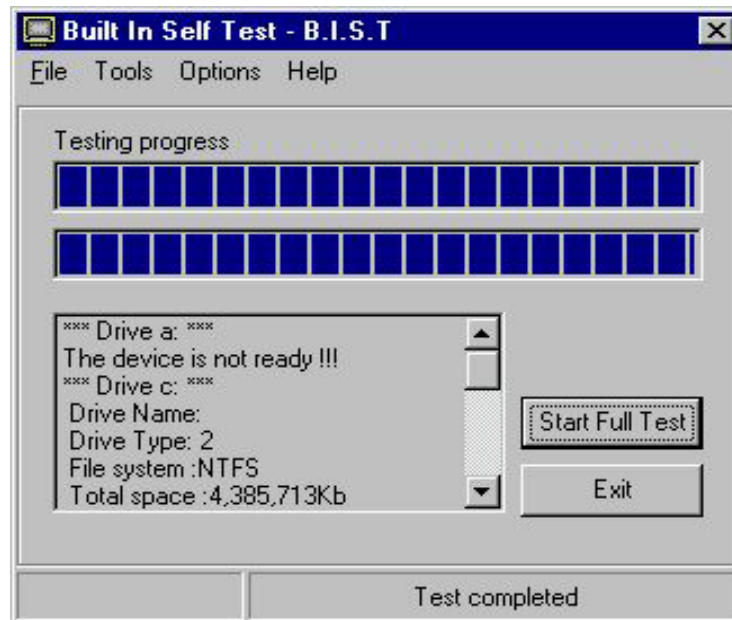


Figure 3 – The BIST module’s graphical interface

The commands are expected in ASCII format, on TCP/IP socket no. 2811. The program will receive commands and will execute the known ones. As a measure of security, if the module receives an unknown command it will send it back.

If the command is correct the BIST module will execute the corresponding test and will send back the results on the same port on which the command arrived.

The BIST module can be used also as stand-alone, through its graphical user interface.

2.2. *The script*

In order to exemplify the concept, we present parts of a PHP script that creates the form, receives the selection from the user, connects to the BIST module, sends the test selections and returns the results. For the sake of simplicity, we work with one BIST module of the DBIST system, i.e. we omitted the selection of the device who’s BIST to run.

The code for the form is simple and will not be presented. We focus on the BIST connectivity part.

The form, as it is displayed in the browser:

Test configuration

BIST module address:

Requested test (case sensitive):

We try to connect to the BIST module at the address specified in the returned form, port 2811:

```
$fp = fsockopen ($addr, 2811);
```

If connection was successful, send the user-requested test name to the BIST module, all uppercase:

```
fputs ($fp, strtoupper($test).".");
```

Receive the test result from the BIST module, maximum 128 characters; append to the result string as we receive, until the transmission end character (".") is met; the "." (dot) operator is string concatenation in PHP (which is, coincidentally, the same as the end-of-transmission character we chose):

```
$recvd = fgets ($fp,128);  
while (!preg_match("/(\.)/",$recvd))  
    { $recvd = $recvd . fgets ($fp,128); }
```

Put the received result into the generated web page:

```
echo "<p><big>$recvd</big></p>";
```

Close the connection to the BIST module:

```
fclose ($fp);
```

Here are the results of the "availspace c:" test:

Test results

BIST module address: **127.0.0.1**

Requested test: **availspace c:**

Opening connection to BIST module...done.

Sending test command "availspace c:"...done.

Reading response from BIST module...done.

Available space :1,188,384Kb.

Closing connection to BIST module...done.

3. CONCLUSIONS

In this paper, we presented a DBIST architecture that uses PHP scripts to let the user select the tests, transmit the selections to the BIST modules and give the test results back to the user. A simple example has been detailed to illustrate the concept.

PHP offers high flexibility, and rapid development, since all that is required from the BIST modules is TCP/IP and ASCII-based communication. Furthermore, PHP is very similar to C, so the learning curve for senior developers is fast, offering a convenient way for Web-enabled DBIST management. The possibilities are wide.

On the other hand, since PHP is an interpreted language, it is slower than native, compiled/linked binaries. But the speed overhead is not in the scripts, but in the BIST modules themselves; however, even these BIST routines are rarely called, compared to the normal functioning of the device. Another limitation may be the need for a web server, although tiny embedded web servers are becoming a one- or two-chip commercial product.

Security has not been implemented, a future development must be to authenticate the user.

4. REFERENCES

1. L. Miclea, Enyedi Sz., R. Orghidan, (2001), "*On-line BIST Experiments for Distributed Systems*", IEEE European Test Workshop ETW'2001, Stockholm, Sweden, May 29th–June 1st, , pp. 37-39.
2. L. Miclea, Enyedi Sz., H. Vălean, (2000), "*Remote Data Acquisition and Control Using Programmable Structures*", Proceedings of International Conference on Quality, Automation and Robotics Q&A-R 2000, Cluj- Napoca, Romania, 19th–20th May, 2000, Tome 2, pp. 425-430.
3. Monica Lobetti Bodoni, A. Benso, S. Chiusano, G. di Natale, P. Prinetto, (2000), "*An Effective Distributed BIST Architecture for RAMs*", Informal Digest of IEEE European Test Workshop ETW 2000, pp. 201-206
4. R. Pendurkar, A. Chatterjee, Y. Zorian, (1996), "*A Distributed BIST Technique for Diagnosis of MCM Interconnections*", International Test Conference 1996 Proceedings, pp. 214-221
5. Y. Zorian, H. Bederr, (1996), "*Designing Self-Testable Multi-Chip Modules*", European Design and Test Conference 1996 Proceedings, pp. 181-185
6. ***, "*PHP Manual*", (2002), <http://www.php.net/docs.php>