AN ARCHITECTURE INDEPENDENT TEST METHODOLOGY FOR SRAM FPGAs

G. S. Bhullar *, V. Szware** and T. A. Kwasniewski*

* Carleton University, Department of Electronics, Ottawa, Ontario, K1S 5B6.
** Communications Research Centre, Ottawa, Ontario, K2H 8S2.

ABSTRACT

The effective utilization of FPGAs in the implementation of reconfigurable circuits is contingent on the logic and routing integrity of the devices. To assure this integrity, a means of detecting and localizing logic and routing resource faults is required. This paper presents an architecture independent methodology for testing logic and routing resources of SRAM FPGAs. The proposed methodology employs a functional level device model for the implementation of architecture independent algorithms that allow dynamic generation of test configurations. The flexible device model allows integration of logic and routing resource testing while enhancing the applicability of the test methodology to a broad range of device architectures and sizes.

1. INTRODUCTION

Current technological advances in SRAM FPGAs have prompted research in reconfigurable systems. For such applications, testing of FPGAs should be extended beyond functional verification of a particular circuit to the total operability of the logic and routing resources. Furthermore, FPGA testing should include fault localization to avoid usage of faulty resources during reconfiguration.

FPGAs are normally vendor tested. However, the failures of concern here are those that occur during integration, system burn-in, or in the field. Field testing differs significantly from the manufacturing test phase. The manufacturer has complete knowledge of device architecture and physical implementation and can map a set of configurations onto the device so as to exercise all device resources. Testing carried out by users is limited to the testing of the manufacturer's functional level device model.

The following sections address the test methodologies for logic and routing resource testing. The implementation and integration of test methodologies through the use of a common device model is described and results are presented.

2. LOGIC RESOURCE TEST METHODOLOGY

The logic resources in coarse grained SRAM FPGAs are composed mainly of the Configurable Logic Blocks (CLBs) which consist of high level primitives such as Look Up Tables (LUTs), multiplexers, and D flipflops. The LUTs and D flipflops can implement combinational and sequential functions, respectively, whereas the multiplexers are mainly used to configure the data paths within the CLBs for implementing the required function. To test the CLBs, it is necessary to exercise all possible modes in which the CLB can be configured.

The test methodology takes advantage of the reconfigurability of SRAM FPGAs to achieve the above aim. The methodology aims at mapping specialized test configurations onto the CLB and exercising it with test vectors to verify the functionality of underlying resources. A CLB test configuration is defined as a circuit which, if mapped onto the CLB, will allow one or more primitives to be exercised with a set of test vectors. The principal guidelines in implementing CLB test configurations are as follows:

1. The CLB test configuration Cn should be such that, for each primitive n with test vector Tn, there should exist a corresponding test set Tn* that will allow Tn to be applied to that module's inputs by applying Tn* to the primary inputs.
2. For each test set applied to a primitive, the test results at the outputs of the primitive should be propagated to the CLB outputs.
3. All primitives must be tested completely with a minimal number of configurations.

The first two objectives imply that inputs of the primitive under test should be controllable from primary inputs and outputs should be observable at primary outputs. The CLB test configurations for the Xilinx XC3000 and XC4000 architectures were developed with these guidelines in mind. It was found that the XC3000 CLB required a minimum of 7 configurations to exercise the entire functionality, whereas the XC4000 CLB required a minimum of 5 test configurations to exercise all functionality other than the carry logic. The smallest device in the XC3000 family is XC3020 with 64 CLBs.
and the largest is XC3195 with 484 CLBs. Clearly, to minimize the overall test requirements, a method is required that will enable the simultaneous testing of a large number of CLBs.

2.1 AN ARCHITECTURE INDEPENDENT METHODOLOGY FOR TESTING MULTIPLE CLBs

To effectively test multiple CLBs, it is necessary to develop a device configuration that allows CLBs to be exercised with a particular CLB test configuration while ensuring that all CLBs under test are controllable and observable. This section describes an architecture independent methodology that allows dynamic generation of device configurations. The methodology incorporates an algorithm that operates on the CLB test configurations and generates netlists for entire device configurations that exercise multiple CLBs. The algorithm is central to the implementation of the methodology and is described below.

The algorithm operates on the given CLB test configurations and a list of available I/Os with the objective of configuring maximum CLBs with a given CLB test configuration while meeting constraints on routing resources and available I/O pins. The upper limit on number of CLBs that can be configured is determined by the number of available output pins and is equal to the total number divided by the number of output pins required by the given CLB test configuration. However, a design with maximum number of CLBs configured may prove to be unroutable. At this point, an iterative process is initiated during which decreasing numbers of CLBs are configured until a routable configuration is feasible. The algorithm then outputs the netlist of the routed device configuration. This cycle is repeated for each CLB test configuration.

This method is particularly suited for devices with boundary scan where the status of I/O pins can be read through the Test Access Port, thus reducing the I/O pin requirements for testing. The above algorithm was implemented as prototype "CLB-Test" software and tested for the Xilinx devices. Section 2.2 presents the results from those tests.

2.2 IMPLEMENTATION RESULTS

"CLB-Test" was tested on the smallest and largest devices in the XC3000 and the XC4000A architectures demonstrating approach extensibility. The devices used in the test included: XC3020, XC3195, XC4002A and XC4005A. For each of the devices, "CLB-Test", was used to generate device configurations for a given CLB test configuration. In addition, two conditions were imposed on "CLB-Test" for the XC3020 device: the availability of all device I/Os for test purposes and a limitation of available I/Os to 48. Table 1 shows the number of CLBs employed by each device configuration for all four devices. The table also shows the actual number of device configurations required to exercise all CLBs with the given CLB test configuration, as opposed to the minimum number possible with the available I/Os. The CLB test configuration chosen for the XC3000 devices required 7 inputs and 2 outputs, whereas for the XC4000A devices required 13 inputs and 3 outputs.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>XC3020</th>
<th>XC3195</th>
<th>XC4002</th>
<th>XC4005</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLB No.</td>
<td>64</td>
<td>48</td>
<td>64</td>
<td>196</td>
</tr>
<tr>
<td>I/O No.</td>
<td>64</td>
<td>48</td>
<td>176</td>
<td>60</td>
</tr>
<tr>
<td>Max. Configurable CLBs per device configuration</td>
<td>28</td>
<td>20</td>
<td>84</td>
<td>15</td>
</tr>
<tr>
<td>Min. Configurations Possible</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>CLBs Placed Per Configuration</td>
<td>484</td>
<td>484</td>
<td>484</td>
<td>484</td>
</tr>
</tbody>
</table>

Table 1: CLB coverage by "CLB-Test" generated device configurations for Xilinx FPGAs.

As observed from the table, the number of CLBs that can actually be configured at a time are much less than the maximum possible. This results from the fact that the routing resources cannot accomodate the large number of nets required for the device configuration. However, the decrease in configured CLBs does not affect the total number of device configurations significantly.

3. ROUTING RESOURCE TEST METHODOLOGY

A fixed design can be tested by means of test vectors which exercise the logic and routing resources used. However, for applications that employ reconfiguration, full testing of the routing resources is desirable. Thus, routing resources other than those used in a particular design must be tested. Furthermore, tests other than those used for logic resources must be employed as the latter do not provide adequate information pertaining to the localization of routing faults. For this reason, routing resource testing must be conducted independently of logic resource testing.

Fault detection and localization in routing resources is carried out by configuring the devices with...
test designs, referred to as configurations, and applying test vectors specific to those designs. Routing resources in an FPGA can be effectively tested by interconnecting the resources to form a path between I/O pins and performing a continuity test on the path. However, the large number of possible paths that the routing resources can be configured in makes it difficult to manually develop configurations that exercise all routing resources. The proposed methodology aims at generating the test configurations dynamically through the use of algorithms specific to each resource type such as local or global lines, switch matrices, and Programmable Interconnect Points (PIPs). The algorithms operate on the routing resource information provided by a functional level device model to generate a test configuration incorporating as many test paths as possible with the given number of available I/Os. The use of the functional level device model enables the algorithms to be architecture independent. The algorithms have been implemented and the resulting software "Routing-Test" has been tested with a fault simulator developed for this purpose. An overview of the test procedure is presented below followed by the results of the tests.

3.1 AN OVERVIEW OF TEST OPERATION

The algorithms presented in previous sections generate configurations for testing various device resources. These test configurations consist of routed designs. To configure the device under test, the routed designs must be converted to the configuration bit-stream for the device and this was done with vendor supplied CAD tools.

Since the design is fully routed and all logic blocks are configured, the design is simply converted to the internal format of the CAD tools for processing by other downstream tools. Most FPGA CAD tools provide a chip editor for directly editing the design on chip. The scripting interface provided by this tool is used to convert the design to the internal format of the CAD tools. A routine has been developed that reads the test configuration and generates a script readable by the chip editor. The script creates all components and nets required by the design while maintaining complete control over the device resources used by the test configuration. Once the design is converted into internal format of the CAD tools, it can be processed by downstream tools to convert it into a configuration bit-stream format suitable for down loading to the device.

The testing of the device is carried out with the help of an IC tester (IMS XL100). The test routine in progress initiates a batch process in the IC tester to download the configuration bit-stream into the DUT and to apply corresponding test vectors to the device. The outputs are sampled and compared with the expected outputs. The fault information thus generated is returned to the test routine for the fault tracing algorithm to process. The various steps in the test sequence are listed below.

1. Test algorithms for various resources operate on the device model and generate a netlist which is converted into a routed design using the router.
2. The routed design is converted into a script readable by the chip editor which is then used to convert the routed design into the vendor proprietary format. The vendor supplied bit-stream generator is used to convert the design to the configuration bit-stream.
3. The IMS tester under the control of the test routine downloads the configuration bit-stream into the DUT. The tester then applies test vectors for that particular configuration to the DUT and compares the outputs with the expected outputs. The fault information generated is given to the fault tracing algorithm.
4. The fault tracing algorithm receives a list of errors observed at the primary outputs and traces these faults to their originating resources, thus localizing faults.

3.2 RESULTS

A fault simulator has been implemented to verify the local line test algorithms. The simulator is capable of modeling single and multiple stuck-at and stuck-open type routing resource faults. The algorithms for local line testing have been tested on the XC3020 device model which has 839 local line segments. In the case of no faults, complete local line testing is achieved in under 10 configurations. For tests with an injected fault set of 18 faults including stuck-at and stuck-open type, all faults were detected and localized within 25 configurations, with no spurious errors detected.

To observe the effect of available I/O pins on the performance of the algorithms, simulations were performed with 16 I/O pins prohibited from use. The simulation results indicate an increase in the number of configurations required to localize all faults by 10. However, the number of configurations in the case of no faults remained unaffected. From the results we conclude that the algorithms perform better with large number of available I/O pins. This requirement is easily satisfied in most devices with boundary scan where all available I/O pins can be used for testing.

4. TEST METHODOLOGY INTEGRATION

The logic resource test methodology introduced in Section 2 allows both the detection of faults in CLBs and the localization of a faulty CLB. However, a fault that manifests itself at a primary output of the FPGA under test may, in fact, be caused by a faulty routing
resource present in the nets connecting primary device I/Os to CLB I/Os. As a result, fault free CLBs may be diagnosed with spurious faults. Such spurious faults can be avoided if fault information from routing resource testing is incorporated during CLB testing. During testing of a particular resource, other resources included in a test configuration may also be tested. If the fault status of these resources is retained and used during the development of test configurations for other resources, the incidence of spurious faults can be reduced. In addition, the fault information from different test phases available through the integration of test methodologies can be cross-examined to improve fault localization. For example, a set of resources with suspected faults may be narrowed if certain resources within the set are found to be fault-free during test phases for resource types other than those in the given set.

From the above discussion, it can be noted that the principal factor towards achieving integration is the availability of fault information obtained from each test phase during generation of subsequent test configurations. Apart from the fault information, it is equally important to have information regarding the use of resources in previous test configurations. The device model employed by the routing resource test methodology stores the above information for use by the test algorithms. The following section addresses the efficient implementation of an integrated test methodology based on a common device model.

5. AN INTEGRATED TEST METHODOLOGY

The device model employed by the routing resource test methodology includes all elements essential for the implementation of an integrated test methodology and can, therefore, be used with minor modifications. Furthermore, the routing resource test algorithms incorporate the fault information from the device model while generating the next test configuration. Thus, these algorithms can be integrated without modifications. However, the algorithms employed by the logic resource test methodology described in Section 2 are not based on the device model, and do not incorporate the fault information obtained during routing resource testing. The problem is thus one of modifying the logic resource test methodology to incorporate the fault information from the device model while generating the next test configuration.

As discussed in section 2, the algorithms generate a netlist for each test configuration, which is routed with the help of the vendor's CAD tools. The use of the router provided by these tools imposes certain limitations, implying that the fault information obtained through prior test phases cannot be used during the generation of subsequent test configurations. Moreover, it implies that the status of the resources used in each test configuration cannot be retained. Thus its use violates the main criterion for integration.

The above problems can be eliminated through the use of a router based on the device model used by the routing resource test methodology. The router includes a provision for taking the fault information in the device model into consideration while generating test configurations. The use of the router allows the incorporation of the fault information from routing resource testing into logic resource test methodology, thus allowing integration of the two methodologies. Such a router has been implemented for the routing resource test methodology.

The use of a common device model allowed the implementation of “FPGA-Tester”, a prototype test software incorporating both logic and routing resource test methodologies. The FPGA-Tester software interfaces to the vendor CAD tools and a generic IC tester enabling application of the test methodology to devices under test.

6. SUMMARY

The testability of logic and routing resources in SRAM FPGAs were presented in terms of two architecture independent methodologies. The methodologies support dynamic generation of test configurations to enhance their applicability to a broad range of device architectures and reduce recurring effort in development of test configurations for different architectures. The algorithms employed by the two methodologies have been implemented and verified and test results based on the prototype software have been presented. Finally, a method has been proposed to integrate the logic and routing resource test methodologies to improve fault detection and localization. The integrated test methodology, based on a common functional level device model and efficient sharing of fault data, has been implemented and the prototype software verified.

REFERENCES