A new automatic method for testing interconnect resources in FPGAs based on general routing matrix

Zhen Yang, Chuanzeng Liang, Jian Wang, and Jinmei Lai

Abstract: Testing of interconnect resources is one of the most important parts in FPGA testing, since most of the transistors in the chip are dedicated to interconnections. Conventional testing methods are no longer as efficient as before because of the various new types of interconnections and a lack of enough input output blocks (IOBs) in the FPGAs that are based on general routing matrix (GRM). This paper presents a new automatic method for testing FPGA interconnect resources in GRM-based FPGAs. This new method, testing line segments and programmable interconnect points (PIPs) in different stages, is applicable to all kinds of GRM-based FPGAs. Experimental results show that a total of 152 test configurations are sufficient to achieve 99.2% and 99.7% fault coverage for line segments and PIPs in Xilinx XC4VLX15 FPGA respectively and to largely reduce the number of IOBs required in the testing.

Keywords: FPGA, interconnect resources, general routing matrix, automatic testing, fault coverage

References

[1] P. H. W. Leong: Electronic Design, Test and Applications (2008) 137. DOI:10.1109/DELTA.2008.14
[2] Z. Marrakchi, H. Mrabet, U. Farooq and H. Mehrez: Int. J. Reconfigurable Comput. 2009 (2009) 6. DOI:10.1155/2009/259837
[3] A. Ruan, W. Tian, B. Ni and K. Wu: Integrated Circuits (2014) 111. DOI:10.1109/ISICIR.2014.7029438
[4] M. B. Tahoori and S. Mitra: IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 24 (2005) 1774. DOI:10.1109/TCAD.2005.852452
[5] M. B. Tahoori and S. Mitra: VLSI Test Symposium (2003) 134. DOI:10.1109/VTEST.2003.1197644
[6] M. B. Tahoori and S. Mitra: Test Conference (2002) 608. DOI:10.1109/TEST.2002.1041812
[7] M. B. Tahoori and S. Mitra: IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 23 (2004) 261. DOI:10.1109/TCAD.2003.822112
[8] M. Renovell, J. Figueras and Y. Zorian: VLSI Test Symposium (1997) 230. DOI:10.1109/VTEST.1997.600278
1 Introduction

Field programmable gate arrays (FPGAs) are integrated circuits consisting of programmable resources. Compared with application specific integrated circuit (ASIC), FPGA has several advantages including lower non-recurrent engineering (NRE) costs, reduced development time, easier adjustment to different communication protocols and reduced risk [1].

Interconnect resources are important parts in FPGAs and usually account for more than half of the total area. In some modern FPGAs, this number even goes up to 90% [2, 3]. Logic blocks and embedded blocks inside an FPGA are usually connected through these interconnections including programmable switches and buffers which make these separate blocks work together properly. It is rather challenging as well as necessary to test all the interconnect resources in an FPGA [4, 5, 6, 7, 8, 9, 10]. Interconnect resources include line segments and programmable interconnect points (PIPs). These resources are vital to the functionality of FPGAs and thus testing them is pretty important. Generally, the exact users’ way to configure the chip is not known during the test, so those interconnect resources must work properly under any user configuration, which is also known as application-independent testing [4, 5, 6, 7, 11]. Since the FPGA test time is dominated by the number of test configurations that are employed, the ultimate goal of interconnection testing is to detect maximum faults with minimum test configurations and to keep the adaptivity to different kinds of FPGAs at the same time [5, 6, 7, 11].

Nowadays, as FPGA interconnection contains much more abundant line segments and variable parameters, traditional connection box and switch box (CB and...
SB) model [12] have been unable to describe the increasingly complex wire connections. As a result, the interconnection architecture based on general routing matrix (GRM) comes into being for its flexibility, high performance and efficiency, thus becoming popular in commercial devices [13, 14, 15, 16].

An automatic test configuration generation strategy of divide and rule targeting FPGA interconnections that are based on PIP’s directions was proposed and investigated in [4, 5, 8]. In these methods, all the PIPs were categorized into four directions, namely horizontal, vertical, left diagonal and right diagonal. These PIPs were further classified into 1-1 mapping and 1-N mapping and tested in two different ways respectively. However, this kind of method is generally applicable to previous simple interconnection structure such as Xilinx 4000 and cannot be applied to FPGAs that are based on GRM, such as Virtex-II, Virtex-4, Virtex-5, etc, where there are a lot of PIPs that cannot be classified into any of the four directions. This problem will be explained in detail in Section 2. Furthermore, GRM-based FPGAs usually contain several other types of routing matrices such as input routing matrix (IRM) and output routing matrix (ORM) [13, 14, 15, 16], both of which contain lots of special PIPs and will undoubtedly add to the difficulty in interconnect testing based on PIP’s directions. In addition, those direction-based methods usually require enough input output blocks (IOBs) to be located around FPGAs, which usually will not be satisfied in modern FPGAs with a huge number of wires and switches.

To address the problem concerning lack of IOBs, a method of built-in self-test (BIST) was proposed in [9]. The BIST method has the advantage of allowing the chip to test itself without any input signals. In a typical BIST method, blocks under test (BUTs) and output response analyzers (ORAs) or test pattern generators (TPGs) exchange positions for many times, which leads to a sharp rise in the number of test configurations. When it comes to GRM-based FPGAs, the TPG in the BIST testing usually has to drive a lot of load, which will also contribute to the difficulty in placement and routing as well as a poor adaptivity to different kinds of interconnect schemes.

Another way to detect faults in FPGA interconnections is based on iterative logic arrays (ILAs), which repeatedly reconfigures the FPGA as a group of testable ILAs [10]. In this way, test configurations can be reduced due to the regularity of the FPGA structure. However, because of a much more complicated structure in GRM-based FPGAs, every logic cell of ILA has to be designed based on its unique structure, which means a lack of flexibility to different kinds of FPGAs and more manual work in generating test configurations.

In this paper, we present a new automatic application-independent testing method for interconnect resources in GRM-based FPGAs, including both automatic test configuration generation and automatic testing system. The novelty of this method is that it is applicable to all kinds of GRM-based FPGAs regardless of the scale of chips. It can also reduce the number of test configurations as well as required IOBs and simplify the testing process with a fully automated testing flow. Experimental results show that this method can cover about 99.7% of the faults in about 150 test configurations for Xilinx XC4VLX15 and XC4VLX200 FPGA.
The rest of this paper is organized as follows. Section 2 introduces interconnect resources as well as fault models in GRM-based FPGAs. Section 3 describes the configuration-generation strategy and algorithms in detail. Section 4 shows the testing flow used in our method. Section 5 presents the experimental results. The paper is summarized and concluded in Section 6.

2 Interconnect resources in GRM-based FPGAs

In a typical GRM-based FPGA, signals can be connected to the inputs of logic blocks through IRM, while signals from the outputs of logic blocks can be connected to global interconnect network through ORM. Both the IRM and ORM can be connected through GRM to the global interconnect system [13, 14, 15, 16]. Fig. 1 shows the basic structure of GRM-based FPGA. The model of interconnect resources discussed in [4] is illustrated in Fig. 2(a). Line segments (dashed lines) are connected through PIPs (bold lines), of which the connectivity is controlled by the values of configuration memory. As is shown in [4, 5, 8], PIPs can be classified into four directions. For example, PIP (W2, E2) is horizontal, PIP (N1, S1) is vertical. PIP (N1, W2) is left diagonal and PIP (N2, E2) is right diagonal. However, two kinds of special PIPs that cannot be classified into any of the four directions have emerged in GRM-based FPGAs. Fig. 2(b) depicts a GRM containing these two special PIPs. One is PIP (E1, E3) which begins and ends at the same side of the GRM. The other one is PIP (W1, S3) (dotted lines), where V stands for a virtual line segment that does not really exist (This is quite common in GRM-based FPGAs [13, 14, 15, 16] where some PIPs just turn to another side of GRM without connecting to a line segment). In this case, PIP (W1, S3) can only be tested when PIP (W1, V) and PIP (V, S3) are connected at the same time, which differs from those single PIP in Fig. 2(a). Note that most of the PIPs in GRM-based FPGAs are based on multiplexers (MUXes) instead of pass transistors, so PIPs in Fig. 2(b) are unidirectional.

Due to a huge number of the special PIPs in GRM-based FPGAs, conventional testing methods based on PIP’s directions to cover all the resources are not as efficient as before. In section 3, we present a new method that solves the problem perfectly.

Fig. 1. The basic structure of GRM-based FPGA.
Faults in FPGA interconnections can be classified into four groups, including open fault of line segments, bridging fault between two line segments, PIP stuck-closed fault and PIP stuck-open fault [17]. Fig. 3 shows the four kinds of different faults. When a PIP stuck fault occurs, it will be permanently open or closed regardless of the value of the memory cell controlling the PIP.

![Fig. 2. Two models of FPGA interconnect resources.](image)

![Fig. 3. Four kinds of faults in FPGA interconnections. (a) Open fault of line segment. (b) Bridging fault between two line segments. (c) PIP stuck-closed fault. (d) PIP stuck-open fault.](image)

3 Automatic configuration-generation strategy

In order to detect all the faults in FPGA interconnections, a graph model that describes the distributions and connections of FPGA interconnect resources should be generated automatically [5]. In this graph, line segment and logic block input or output pin are represented with nodes, unidirectional switch (such as PIP) is represented with a directed edge and bidirectional switch (such as pass transistor) is represented with a pair of directed edges. When it comes to large scale FPGAs, other problems concerning the storage of the graph need to be dealt with, which is beyond the scope of this paper and detailed explanation to these problems can be found in our previous work in [18]. Fig. 4 gives a simple example of the graph model corresponding to the GRM in Fig. 2(b). When it comes to special PIPs (e.g. the point of V in Fig. 4), we describe them in the same way as common PIPs in the graph except that these new PIPs will never be used as an ending point of a wire under test (WUT) since they do not really exit as explained in section 2. Since they are described in the same way as common PIPs, these special PIPs can also be tested in the our methods. Besides, the presented methods and algorithms in this paper can be easily applied to other FPGA interconnect schemes because of the versatility of the interconnection graph model.

The most important part of FPGA interconnect testing method is to detect the maximum number of faults with minimum number of configurations. In this section, a fully automated testing method is presented. All the interconnect
resources are categorized into two groups, namely line segments and PIPs, which are tested in stage 1 and stage 2 separately. At last, stage 3 is performed to further improve the fault coverage.

3.1 Stage 1
Stage 1 is designed to detect possible faults in line segments. In stage 1, WUTs are created in an S-type way. An S-type WUT begins with an IOB and connects each line segment one by one. When run into the end of every row or column, the S-type WUT turns into adjacent row or column instead of ending. This process will be repeated until the IOB in the last row or column is reached. Obviously, the length of an S-type WUT is the number of array rows multiplied by the number of columns while the length of horizontal-only WUT is just the number of columns. Take Xilinx Virtex-4 XC4VLX200 FPGA as an example, every WUT between a pair of IOBs is 56,854 (217 rows \times 262 columns), compared with 262 for horizontal-only WUT and 217 for vertical-only WUT. It is easy to find out that the S-type WUT can cover much more lines and PIPs than the horizontal-only or vertical-only WUT between two IOBs, as is shown in Fig. 5. In this way, both the number of IOBs required and the number of test configurations are reduced significantly compared with those required in [4].

3.2 Stage 2
Stage 2 is designed to detect the possible faults of PIPs. In this stage, an enhanced depth first search (DFS) algorithm based on the PIP’s weight is adopted. DFS is an algorithm for traversing or searching tree or graph data structures. One starts at the root and explores as far as possible along each branch before backtracking. DFS is quite suitable to cover PIPs in stage 2 because the paths generated by DFS tend to be quite long and complicated thus involving as many PIPs as possible and reducing the number of IOBs required in the test. However, when it comes to testing, we not only need to visit all the nodes and edges in the graph model, but also have to create a complete WUT containing the minimum number of the PIPs that have been tested in other WUTs. In this paper, we set a parameter of weight for PIPs (i.e., edges in the graph model). A PIP’s weight is defined as follows:

![Fig. 4. The interconnect graph model corresponding to the interconnections in Fig. 2(b).](image)
Weight(\(n\)) = h_0 \times f(n) + (1 - h_0) \times g(n)  \tag{1}

where \(n\) denotes an edge in the graph model, \(f(n)\) denotes the usage of a certain edge and gets larger when the edge is repeatedly used. \(g(n)\) is determined by the Manhattan distance between the node reached through the edge and the node at the end of the WUT, a larger distance will lead to a smaller \(g(n)\). In order to cover more resources in a single configuration, a longer distance (i.e., a smaller \(g(n)\)) and a longer WUT are preferred. \(h_0\) is a constant between 0 and 1 to adjust the trade-off between \(f(n)\) and \(g(n)\). Experimental results show that when \(h_0\) equals to 0.83, the algorithm gets the highest efficiency. It works in such a way that the more times a PIP is used, the larger its weight will be. In the enhanced DFS algorithm, every time a new WUT is created, PIPs with the lightest weight will have the highest possibility to be involved in, thus getting the least PIP overlap between different configurations and reducing the number of test configurations greatly. Fig. 6 describes the enhanced DFS algorithm based on PIP’s weight.

1. create graph \(G(V, E)\) for interconnect resources
2. \(V = \{\text{line segments and pins of logic blocks}\}\)
3. \(E = \{\text{common PIPs}\} \cup \{\text{special PIPs}\}\)
4. for each edge \(e\) in \(E\) do
5. \(W_e \leftarrow \text{weight of } e\)
6. add source and sink vertices \(s\) and \(t\) to \(G\)
7. \(Q \leftarrow \text{empty priority queue}\)
8. repeat
9. set \(s\) visited
10. push all child vertices of \(s\) into \(Q\)
11. select a child \(c\) of \(s\) with minimum weight
12. \(s \leftarrow c\)
13. delete \(c\) from \(Q\)
14. until \(Q\) is empty

Fig. 5. An S-type WUT and a horizontal-only WUT.

Fig. 6. The enhanced DFS algorithm based on PIP’s weight.
3.3 Stage 3

Stage 3 is designed to compensate for the decrease of the efficiency in Stage 2 after most of the PIPs are covered and to further improve the fault coverage by a customized algorithm aimed at PIPs that have never been tested in Stage 1 or Stage 2. At the beginning of Stage 2, all the PIPs have never been tested and each configuration will contain enough new PIPs. However, as more and more PIPs are tested after several configurations in Stage 2, the number of untouched PIPs will decrease significantly, leading to a fall in the amount of new PIPs in the configurations generated afterwards which can be seen in Fig. 7 (dashed line). Generally, at the end of Stage 2, most of the untouched PIPs tend to be located in the corner of the chip, which are close to IOBs and lack a good connectivity to other PIPs. To address this problem, in Stage 3, a WUT is created only when the number of new PIPs it contains exceeds a certain threshold. In this way, test configurations generated in stage 3 will contain enough untouched PIPs, thus getting the maximum efficiency of each single configuration. Fig. 7 shows the comparison between the number of new PIPs in the configurations that are created in Stage 2 and Stage 3 respectively. As can be seen, after the 137th test configuration is created, if Stage 2 is continued, new PIPs contained in the next configurations will decrease greatly. This problem will be relieved when Stage 3 is performed, which keeps a high efficiency of configurations afterwards. A common breadth first search (BFS) algorithm is used in search of the untouched PIPs in Stage 3.

![Fig. 7. Comparison of new PIPs contained in the configurations created in Stage 2 and Stage 3.](image_url)

4 Automatic FPGA testing flow

In order to verify the method presented in this paper, we have designed an automatic testing platform. After all of the test configurations are generated in the three stages mentioned in Section 3, some steps are taken to download the configurations into the chip and to get the final results. Firstly, test configurations are written as text files. Secondly, another text file containing testing vectors and expected responses is attached to each configuration file. To detect the open faults in a WUT, both logic 0 and 1 must be applied. To detect bridging faults between all
WUTs, any two WUTs must be applied with different logic values (e.g., one is logic 0 and the other one is logic 1) at least once. According to [19], to find out all the open and bridging faults within n WUTs, a total of $1 + \log_2 n$ tests are required. Fig. 8 gives an example of the text file containing testing vectors and expected responses for 8 WUTs. The vectors after “I” is the pattern to be applied to each WUT and the identical vectors after “O” is the expected response. Thirdly, bitstream file containing configuration memory values is generated based on the test configurations. Finally, the bitstream files are downloaded into FPGA to configure the chip. In addition, test vectors are applied and responses are observed through a joint test action group (JTAG) interface. All the steps are carried out automatically. If all the responses observed from the end of each WUT are exactly the same as the expected values defined in the testing files in advance, conclusions can be drawn that there are no faults in the FPGA interconnect resources. Fig. 9 describes the flow of FPGA testing.

![Fig. 8. Testing vectors and expected responses for 8 WUTs.](image)

![Fig. 9. FPGA testing flow.](image)
5 Results

We implemented the presented technique consisting of three stages for two GRM-based FPGAs, including Xilinx Virtex-4 XC4VLX15 and XC4VLX200 which are the smallest and largest FPGA respectively in Virtex-4 series. Fig. 10 shows the FPGA testing platform including a computer and a printed circuit board (PCB) containing the FPGAs and a JTAG interface. Following the testing flow in Fig. 9, all the steps were carried out using a C++ program.

![Fig. 10. The FPGA testing platform.](image)

Table I lists the number of test configurations for Xilinx XC4VLX15 and XC4VLX200. Despite the fact that XC4VLX200 is much larger than XC4VLX15, the number of configurations just increased a little bit from 152 to 158 regardless of the scale of the FPGAs. Table II shows the comparison between the number of IOBs needed per test configuration, using methods in [4, 5] and method in our paper respectively. Using our method, the number of IOBs can be reduced greatly. Table III and IV illustrates comparisons of the fault coverage of common PIPs and special PIPs in XC4VLX15 and XC4VLX200, using method in [4, 5] and method in our paper respectively. Common PIPs refer to PIPs that can be categorized into any of the four directions while special PIPs cannot. For common PIPs, methods in [4, 5] can be applied to achieve less than 100% fault coverage. However, the method is unable to cover any special PIP and the fault coverage for those PIPs is zero. Using our method, a total of 99.7% and 99.6% fault coverage is achieved, including both common and special PIPs, and gets more and more efficient as the scale of the FPGAs gets larger and larger. Table V lists the comparison of fault coverage for line segments in XC4VLX15 and XC4VLX200.

| Table I. Number of configurations for XC4VLX15 and XC4VLX200 |
|---------------------------------|-----------------|
|                                | XC4VLX15 | XC4VLX200 |
| Stage1                         | 67       | 69        |
| Stage2                         | 70       | 71        |
| Stage3                         | 15       | 18        |
| Total                          | 152      | 158       |

| Table II. Number of IOBs needed per test configuration |
|---------------------------------|-----------------|
|                                | Methods in [4, 5] | Our methods |
| IOBs                           | >5000           | 8~20         |
6 Conclusion

In this paper, we have presented an application-independent method for testing FPGA interconnect resources in three stages with a fully automated testing flow. Test configurations are generated, downloaded into FPGA and then tested automatically. In this method, line segments are tested in an S-type way and PIPs are tested based on their weights instead of their directions, which brings great flexibility to different FPGA interconnect schemes and reduces the number of IOBs required in the test greatly. In this way, both common and special PIPs in GRM-based FPGA can be tested efficiently. In addition, due to the utilization of the interconnection graph model, all the methods and algorithms presented in this paper can easily be applied to other architectures of FPGAs.

| Table III. Comparison of fault coverage of PIPs for XC4VLX15 between two methods |
|---------------------------------|------------------|-----------------|
|                               | Number of PIPs   | Methods in [4, 5] | Our methods |
| Common PIPs                   | 5,167,224        | ≤100%            | 99.6%        |
| Special PIPs                  | 1,270,087        | 0%               | 100%         |
| Total                         | 6,437,311        | ≤80%             | 99.7%        |

| Table IV. Comparison of fault coverage of PIPs for XC4VLX200 between two methods |
|---------------------------------|------------------|-----------------|
|                               | Number of PIPs   | Methods in [4, 5] | Our methods |
| Common PIPs                   | 14,437,001       | ≤100%            | 99.4%        |
| Special PIPs                  | 5,490,744        | 0%               | 100%         |
| Total                         | 19,927,745       | ≤73%             | 99.6%        |

| Table V. Fault coverage of line segments |
|------------------------------------------|
| FPGAs                                    | Fault coverage  |
| XC4VLX15                                  | 99.2%           |
| XC4VLX200                                 | 99.8%           |