FPGA Interconnect Testing Algorithm Based on Routing-Resource Graph

Li Dai, Zhi-bin Liu, Shao-chi Liang, Meng Yang, Ling-li Wang*
State Key Lab of ASIC & Systems, Fudan University, Shanghai 201203, P. R. China
*Email: llwang@fudan.edu.cn

Abstract

Static-random-access-memory (SRAM)–based field programmable gate arrays (FPGAs) consists of 50%~70% routing resources. A simple programmable interconnect point (PIP) is a switch controlled by SRAM configuration cell connecting two wires. A novel traverse algorithm targeted for the detection of PIP open faults is proposed. Experimental results run on the Fudan Design System (FDS) platform show that the algorithm is effective to examine the open faults of the routing paths caused by the PIPs fault configuration.

1. Introduction

Compared with application-specific integrated circuits (ASICs), the most significant characteristic of FPGAs is their flexible programmability, which helps in achieving a short design cycle and no Non-Recurring Engineering (NRE) costs, as well as a reduced time to market [1]. With these advantages, FPGAs are increasingly used in the applications where previously was an exclusive territory of ASICs [2].

Two factors combine to determine the flexible programmability of an FPGA: the quality of the CAD tools used to map the design into the FPGA and the quality of the FPGA hardware architecture. Generally, SRAM-based FPGAs are composed of configurable logic blocks (CLBs), I/O blocks (IOBs) and programmable routing resources [3]. A PIP is a switch controlled by SRAM configuration cell connecting two wires. By generating different configurations, an FPGA can achieve incredible flexibility to implement any digital circuit on the same piece of silicon [4]. Faults in the FPGA interconnect can be categorized in two major groups, namely opens and shorts. An open fault can be a PIP stuck-open or an open on a line segment. A short fault can be a PIP stuck-closed or a bridging fault between two routing resources. A PIP stuck-open/stuck-closed fault causes the PIP to be permanently open / closed regardless of the value of the memory cell controlling the PIP. According to the fault models, many researches focus on the particularity of the hardware structure and the relevant algorithms development [5]-[8]. These solutions guarantee to achieve 100% fault coverage by using the least configuration files.

To the best knowledge of the authors, no research is reported to study verification of the CAD tools used to map the design into the FPGA. Especially whether each PIP being configured correctly is required to be verified, according to the result after the circuit is routed. This paper presents an algorithm to verify whether PIPs fault configuration exists.

The rest of the paper is organized as follows. Section 2 gives the FPGA routing architecture used in the research as well as background of routing-resource graph and bitstream generation. Section 3 illustrates the proposed algorithm in details. Finally experimental results and conclusions are discussed in Section 4 and Section 5 respectively.

2. Background

2.1 FPGA Routing Architecture

In this paper, the island-style routing architecture is under investigation. Figure 1 depicts an island-style FPGA. Logic blocks are surrounded by routing channels of pre-fabricated wiring segments. A logic block input or output, which we call a pin, can connect to some or all of the wiring segments in the channel adjacent to it via a connection box (CB) of programmable switches. At every intersection of a horizontal channel and a vertical channel, there is a switch box (SB). This is simply a set of programmable switches that allow some of the wire segments incident to the switch box to be connected to other wire segments. All of the programmable switches in the CBs and SBs are controlled by PIPs. As shown in Figure 1, I1 and I2 are the input pins of CLB, while O1 and O2 are the output pins of IOB/CLB. L1-L8 are wiring segments.

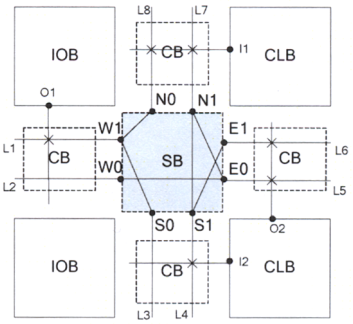

Figure 1. An island-style FPGA

978-1-4244-2186-2/08/$25.00 ©2008 IEEE

2.2 The Routing-Resource Graph

A routing-resource graph (RRG) is used to represent the FPGA routing resources. Each wire and each logic block pin becomes a node in this routing-resource graph and each switch becomes a directed edge (for unidirectional switches, such as buffers) or a pair of directed edges (for bidirectional switches, such as pass transistor) between the two appropriate nodes, where there is a PIP. Figure 2 shows the routing-resource graph corresponding to Figure 1. As shown in Figure 2, there is a directed edge between O1 and L1 because output pin O1 connects to wiring segment L1 through a unidirectional switch. However, a pair of directed edges between L1 and L8 are shown in Figure 2 because a pass transistor is used to connect them.

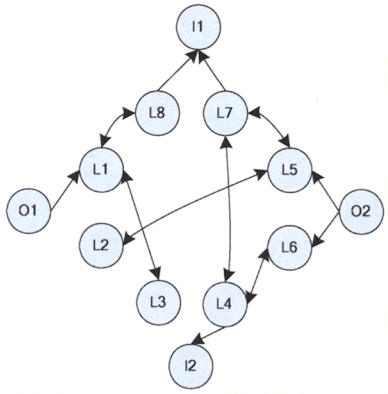

Figure 2. Routing-resource graph corresponding to Figure 1

2.3 Configuration Bitstream Generation

Although the number of the PIPs of the SRAM-based FPGA is very large, a PIP can be uniquely located by the combination of different address line and data line. Hence, chip configuration library can be used to memorize and calculate the exclusive address line and data line of each PIP in the target FPGA.

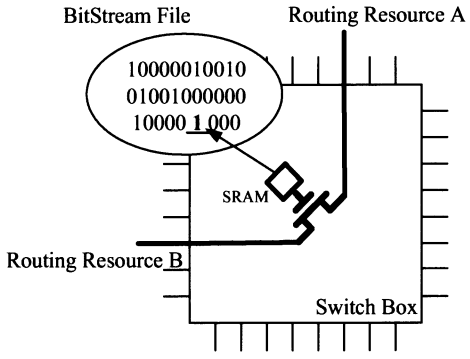

Figure 3. Configuration bitstream

For an SRAM-based FPGA, a PIP in its simple form is a switch controlled by SRAM configuration cell connecting two wires. Figure 3 shows the principle of the configuration bitstream generation. There is a given connection between wire segment A and B with a pass transistor controlled by a PIP. Configuration bitstream generation module then looks for this PIP in the chip configuration library according to the connection, getting the coordinate (address line and data line) of the PIP, in order to ensure the PIP's position in the generated bitstream file. The pass transistor is assumed to be on if the PIP's value is "1"; otherwise it is assumed to be off. Therefore, if wire segment A and B need to be connected together, "1" should be placed at the appropriate position in the generated bitstream file to turn on the pass transistor.

3. Proposed Algorithm

3.1 Overview

In order to validate whether chip configuration library can correctly locate each PIP, the traverse algorithm needs to cover the greatest extension of the PIPs in the target FPGA (PIPs in SBs and CBs). Moreover, an effective testing strategy should be developed so that each output signal can be observed to find out conveniently the routing paths on which any PIP fault exists. Hence it starts with a net connecting to the input IOB as a driver. Next, input signals are added to the net so as to traverse as many routing resources as possible. The traversed path corresponds to the edge in the routing-resource graph. Finally, the input signal carried by the routing paths comes out from the other output IOBs for observation.

In order to transfer the input signal to each IOB, all the PIPs turning on the programmable switches on the given routing paths must be fault-free. When PIPs fail to be configured if a PIP records the wrong address line and data line in the chip configuration library, the corresponding switches will be turned off instead, causing the routing paths to be electrically open. Consequently, some output signals carried by these false paths are no longer equal with the input signal.

Our traverse algorithm is based on this crucial feature, to achieve the goal of identify any fault in the chip configuration library.

3.2 Modification of Routing-Resource Graph

For an FPGA router, each CLB and IOB pin becomes a source/sink of a net in the routing-resource graph. However, this definition is not convenient for the validation of bitstream information. By constructing an appropriate routing-resource graph we can only use IOB pins as sources/sinks, with CLBs configured to act logically as a wire. Hence, the length of the internal net

can be extended. Also, the signal from an input IOB can only end at another output IOBs of the chip, corresponding to the net terminals. As an example in Figure 4, there are eight input pins In1-In8 and two output pins Out1-Out2. The signal carried by each input pin could output to each output pin with corresponding configurations. So in the modified RRG, all the edges (In_i, D_j) are considered as PIP edges. The D1 and D2 are extra virtual vertices. They are added before each Output pin to flag that it connected between the CLB input pins and the CLB output pins.

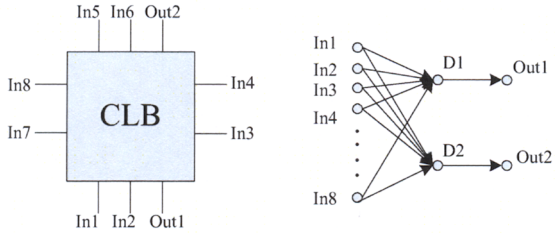

Figure 4. (a) Structure of CLB (b) Equivalent RRG after modification

3.3 Cost Function

To achieve the validation of PIPs, two factors must be considered:

1. each configuration file needs to cover as many PIPs as possible;
2. PIPs that have been used before should hardly appear in the new configuration files, unless a certain net is illegal without these PIPs.

Hence the cost function is defined as in (1).

$$Weight(n) = Crit * UsedWeight(n) + (1\text{-}Crit) * DireWeight(n,sink). \quad (1)$$

where *UsedWeight*(*n*) term is the times that a certain node has been used. The *DireWeight(n,sink)* term is the orientation offset between a certain node "*n*" and the sink node in the same net. *Crit* is a key parameter that controls the trade-off between *UsedWeight* and *DireWeight* for each evaluated node.

The more times a node has been used, the smaller the *UsedWeight(n)* term will be. The larger the orientation discrepancy between node "*n*" and sink node is, the larger the *DireWeight(n,sink)* term will be. The first term (*UsedWeight*) should be added more weight, since the nodes which have been used once should be avoided using again. Moreover, the algorithm can consider choosing the nodes farther from the sink node only if current nodes have the same case. With this consideration, the appropriate value is experimentally determined and set to 0.7.

3.4 Pseudo Code

The pseudo code of the algorithm is given in Figure 5. It should be noted when adding node n's adjacent node m to priority queue, one must ensure node m doesn't have parent node, or there is a loop in the net.

Let: RT(i) be the set of nodes, n, in the current routing of net(i)

```
CreateSourceAndSink(); //according to numbers of
                       //IOBs, make sure the input
                       //IOB and output IOBs,
for (each net with certain source and sinks) {
    add souce pin to RT(i) and PriorityQueue;
    for ( each sink,j, of net(i) ) {
        remove highest weight node, n, from current
        PriorityQueue;
        while( sink(i,j) not found ){
            for ( all fanout nodes m of node n) {
                add m to PriorityQueue at
                PathWeight(m) = Weight(m) +
                PathWeight(n);
            }
            Remove highest weight node from current
            PriorityQueue;
        }
        if( found sink(j) of net(i) )
            backtrace net(i);
        updateWeightNodePriorityQueue();
        //the more close to the source node, the lower
        //weight node
    }
    exportNetFile();
}
```

Figure 5. Pseudo code of the traverse algorithm

4. Experimental Results

The proposed traverse algorithm is coded in C++ under windows and evaluated on Fudan Design System (FDS) platform, in which consists of an FPGA chip FDP250K-II. The specification of FDP250K-II is shown in Table 1.

Table 1. Specification of FDP250K-II

Number of CLBs	Number of IOBs	Number of user IOBs	Number of PIPs
400	144	102	425812

One net file is generated as follows: one of 144 IOBs is selected as an input and the remainder are used as outputs. As can be seen in Table 1, although there are only 102 user IOBs, through JTAG all the 144 IOBs could catch the output signal. In order to cover all the

cases, 144 different net files are generated in the same manner as the generation of one net file. Satisfyingly, each net file achieves about 3% coverage averagely. In order to route all nets successfully, some edges in RRG need to be reused. Figure 6 shows the proportion of edges never used before in each net file. At first, for all the edges could be chosen, the proportion could reach 3.5%. With more and more edges being used, the proportion is decreased. All 144 net files achieve 99.59% coverage of PIPs in FDP250K-II.

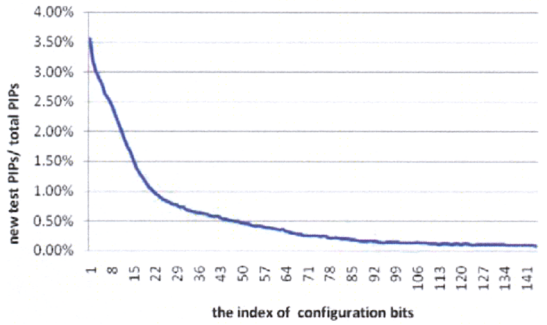

Figure 6. Proportion of new test PIPs to total PIPs in each net file

Figure 7 shows the whole net in a net file. As in Figure 7, all the routing channels in the FPGA are in yellow, showing that the algorithm achieved in covering almost all the routing resources.

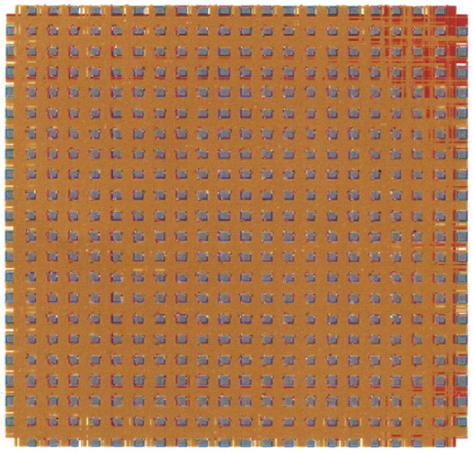

Figure 7. The whole net in a net file

5. Conclusions

A novel traverse algorithm targeted for the detection of open PIP faults is proposed. It uses a routing-resource graph to represent the corresponding FPGA architecture. By efficiently searching the RRG, open faults of the routing paths caused by the PIPs can be identified and the chip configuration library can be verified conveniently. The overall coverage of PIPs of FDP250K-II is up to 99.59%.

Acknowledgments

This project is supported by the National Natural Science Foundation of China under Grant No.60676020.

References

[1] Mehdi B. Tahoori and Subhasish Mitra, "Application-Dependent Delay Testing of FPGAs", IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, vol.26, No.3, pp.553(2007).

[2] Shahin Toutounchi and Andrew Lai, "FPGA Test and Coverage", International Test Conference (ITC), pp.599(2002).

[3] Xilinx, Virtex-5 FPGA User Guide, UG190 (v4.0) March 31, 2008.

[4] Mehdi Baradaran Tahoori and Subhasish Mitra, "Application-Independent Testing of FPGA Interconnects", IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, vol.24, No.11, pp.1774(2005).

[5] Michel Renovell, Jean Michel Portal, Joan Figueras and Yervant Zorian, "Testing the interconnect of RAM-Based FPGAs", IEEE Des. Test. Comput. , vol.15, No.1, pp.45-50(1998).

[6] Charles Stroud，Jeremy Nall，Matthew Lashinsky and Micron Abramovici, "BIST-Based Diagnosis of FPGA Interconnect", International Test Conference (ITC), (2002).

[7] J.Xu, A.Alimohammad and Pieter Trouborst, "Minimal Test Configurations For FPGA Local Interconnects", Proceedings of the 2002 IEEE Canadian Conference on Electrical & Computer Engineering, (2002).

[8] B.Liu, F.Lombardi and W.K.Huang, "Testing Programmable Interconnect Systems: An Algorithmic Approach", Proc. IEEE Asian Test Symp. , Taipei, Taiwan, pp.311-316(2000).