Computing and Informatics, Vol. 21, 2002, 489–505

FAULT-TOLERANT FPGA-BASED SYSTEMS

Khaled Elshafey

System and Computer Engineering Department
Faculty of Engineering, Al-Azhar University
Cairo, Egypt
e-mail: k_elshafey@hotmail.com

Jan Hlavička

Computer Science and Engineering Department
Czech Technical University
Karlovo nám. 13, 121 35 Prague 2, Czech Republic

Manuscript received 23 April 2002; revised 30 October 2002
Communicated by Norbert Frištacký

Abstract. This paper presents a new approach to on-line fault tolerance via reconfiguration for the systems mapped onto field programmable gate arrays (FPGAs). The fault detection, based on self-checking technique, is introduced at application level; therefore our approach can detect the faults of configurable logic blocks (CLBs) and routing interconnections in the FPGAs concurrently with the normal system work. A grid of tiles is projected on the FPGA structure and a certain number of spare CLBs is reserved inside every tile. The number of spare CLBs per tile, which will be used as a backup upon detecting any faulty CLB, is estimated in accordance with the probability of failure. After locating the faulty CLBs, the faulty tile will be reconfigured with avoiding the faulty CLBs. Our proposed approach uses a combination of hardware and software redundancy. We assume that a module external to the FPGA controls automatically the reconfiguration process in addition to the diagnosis process (DIRC); typically this is an embedded microprocessor having some storage for the various tile configurations. We have implemented our approach using Xilinx Virtex FPGA. The DIRC code is written in JBits software tools. In response to a component failure this approach capitalizes on the unique reconfiguration capabilities of FPGAs and replaces the affected tile with a functionally equivalent one that does not rely on the faulty component. Unlike fixed

structure fault-tolerance techniques for ASICs and microprocessors, this approach allows a single physical component to provide redundant backup for several types of components.

Keywords: FPGAs, configurable logic blocks (CLBs), fault tolerance, reconfiguration, redundancy

1 INTRODUCTION

Design of fault-tolerant systems (FTS) is gaining popularity and importance, because high reliability and high availability are becoming a standard requirement. Typical application areas are the transportation, communications, on-line process control, or multiple-access transaction processing. Fault tolerance (FT) can be carried out by means of using spare resources to replace the faulty ones and to take over their function. However, it mostly leads to increased area overhead and degradation in the system performance, but also to the increase in energy consumption, heat dissipation and weight and volume increase. Due to the current progress in semiconductor technology, these drawbacks are becoming less important and are largely overweighed by the desirable functional properties of the resulting systems.

There are many approaches to FTS design that can be classified according to several criteria. To the most important criteria belongs the level of FT tool application. Here we can distinguish three main streams: system-level FT, building block (intermediate) level FT and component-level FT. As the designers mostly cannot change the internal structure of the components used, the first two approaches are used more frequently and are therefore developed to a higher degree of sophistication [17]. This paper, on the other hand, deals with the idea of embedding the FT directly in the components with the goal of designing FTS.

The regular structure of FPGAs and their inherent redundancy make them an almost ideal tool for the implementation of low-level FT technique. Since a typical design uses only a portion of the logic and interconnect resources of the FPGA, the unused resources can provide spares to be used to replace the faulty ones. To use these spares systematically, we must solve the following main problems:

- detection of a failure during the component application,
- location of the fault to within a replaceable unit,
- reconfiguration of the FPGA structure in accordance with the diagnostic data obtained.

These three problems will be treated in the present paper.

2 DIFFERENT APPROACHES TO FPGA FAULT TOLERANCE

Recent advances in fabrication technology and device architecture have resulted in tremendous growth of FPGA density. Currently commercially available devices can map up to one million or more system gates, for instance Xilinx Virtex FPGA series has densities from 50 k to 1 M system gates [20]. Because of their short turnaround time and programmability in the field, they have been widely used for rapid prototyping and hardware emulation of VLSI chips and other purposes. Now they are increasingly used in mainstream applications, such as communications or digital signal processing [3]. One important class are SRAM-based FPGAs, which can easily be reprogrammed any number of times. As FPGAs increase in size and complexity, their probability of faults also increases; therefore thorough testing becomes increasingly important. FPGAs may suffer from a variety of faults. These include: wires having breaks or two wires being electrically connected; a programmable connection being inoperative so that the wires can only be connected or can only be disconnected; and a programmable logic element not capable of being programmed as intended due to various faults. Thus, a testing procedure ensuring reliable reuse and functional modifiability of FPGAs is needed [15].

FPGA FT has been the focus of several recent papers [2, 4, 12]. In [12] the authors proposed a low-overhead FT FPGA system approach that capitalizes on the unique reconfiguration capabilities of FPGAs. The physical design is partitioned into tiles and the tiles are partitioned into subtiles, each containing at least one spare CLB. Tiling provides many advantages in the implementation of fault-tolerant FPGA systems. First, the amount of memory needed to store the set of alternative configurations is smaller than the amount required to store a set of complete configurations. In addition, it increases reliability.

To enforce the fault-tolerance at run-time, all tile configurations are stored in an external memory before the circuit begins its operation. The system normally runs with the original configuration until a fault is detected. Upon detection, the circuit ceases the functional mode until the proper reconfiguration of the faulty tile is activated. The result of this reconfiguration is the elimination of the faulty CLB and its replacement by a fault-free one. However, there are unspecified test and diagnosis mechanisms assumed, the number of spare elements not estimated, and the size of tiles is not specified either.

In [2], a dynamic FT approach for CLBs of an FPGA has been introduced. An on-line testing, diagnosis, and fault tolerance techniques are integrated into a unified framework. The testing is performed in two spare regions (vertical and horizontal) of the chip called self-testing areas (STARs), concurrently with the normal system operation in the rest of the chip as shown in Figure 1. After a STAR is completely tested, it roves across the chip so that eventually the entire FPGA is tested. Roving involves transferring a portion of the system function into the previous STAR and configuring a STAR in the just released region. A STAR consists of several disjoint tiles, which are tested concurrently. Every tile contains a test structure called BISTER. Both the testing process and the STAR roving mechanism rely on run-time

reconfiguration (RTR) and are controlled by a Test and Reconfiguration Controller (TREC) external to the FPGA. TREC is implemented as an embedded processor with means for fault tolerance. TREC reads the test results from every BISTER, and when test failures are reported by a BISTER, TREC initiates diagnostics to locate the faults, and performs the reconfigurations needed to bypass the located faults. However, this approach detects faults in the STAR, so if a fault occurred in the working area, it will not be detected until this area is under test. This approach considers the FPGA at a low level of abstraction, as a matrix of a CLBs interconnected through general routing switches. The specific characteristics of the application circuit mapped onto the FPGA are not taken into account. The fault latency (the interval between the occurrence of a fault and its detection) is bounded by the interval required for testing the entire FPGA.

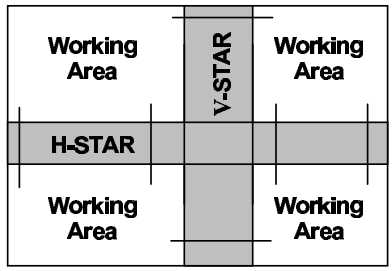

Fig. 1. Moving area under test across the chip

In [4], a high-level approach to the design of fault-tolerant systems by using on-line detection and reconfiguration in FPGA architectures has been introduced. Here the fault detection techniques are introduced at the application system level. The underlying idea is partitioning the dataflow path of the application circuit into disjoint subgraphs in which self-checking is achieved. These subgraphs are then suitably grouped into disjoint clusters, each of which is mapped onto a tile (portion of the FPGA). When a fault is detected in a subgraph (subtile) of the tile the whole tile is reconfigured onto a spare tile of the FPGA without requiring a specific diagnostic phase. The main advantages of this approach are limiting the circuit complexity of the overall system through reducing the number of checking points, as well as reducing the total latency by avoiding a specific diagnostic phase. However, this approach is very area consuming because the whole faulty tile will be reconfigured onto a spare tile in spite of the fact that only a part of the tile (maybe only one CLB) is faulty, another problem is the size of the spare tile. We know that the sizes of the tiles are not equal and they are based on the application circuit. Therefore the spare tile to be used to replace a tile must be large enough to host the largest tile mapped onto the FPGA, and in this case there will be more wasted CLBs.

Our approach is based on a tiling approach as in [2, 4, 12], but with some differences. There is specific testing and diagnosis mechanisms used, the faults are

detected on-line, the number of spare elements is flexible, and the JBits tool is used to control the FT process. The designer partitions the physical circuit into a set of tiles and applies a self-checking technique for each tile. The sizes of the device tiles are equal to the sizes of the application tiles, with some spare CLBs added for the purpose of the FT. We aim to minimize the area overhead by locating the faulty cells inside the faulty tile and using spare cells inside the same faulty tile instead of the faulty ones during the reconfiguration process. The sizes of the tiles will be small because each individual disjoint sub graph including its checker will be mapped onto a different tile and this will lead to the simplicity of the reconfiguration process.

3 FAULT MODEL

The CLB usually consists of three types of logic modules: D flip-flops, multiplexers and look-up tables (LUTs) [20]. Multiplexers and LUTs are typical configurable devices, while D flip-flops are not really configurable, although their asynchronous control signals (reset, clock) are configurable by means of multiplexers. Our approach detects the faults at the application system level, so it can detect CLB faults or the routing switches and wires faults. However, our fault location algorithm can locate faults only in the CLBs, therefore we presume that the CLBs may be faulty, while the switches and wires are fault-free. For instance, LUT faults are such that any number of the configuration bits could be stuck-at or could have address line faults (functional inputs). In our approach, any number of faulty CLBs can be detected. There are two classes of faults, the first containing faults that are independent and identically distributed (iid), i.e., uniformly distributed faults, where each element fails with independent probability. Secondly, there are nonuniform faults (clustering model), which postulate the existence of non-overlapping regions, alternatively known as quadrats or blocks [13]. For our study we will assume the first class.

4 ON-LINE TESTING APPROACH

As the reconfiguration should happen during the application run, we need a reliable tool to discover any failure immediately after its occurrence. A method that may serve this purpose is the self-checking for the given class of faults [1]. This means that using a code checker, we can determine whether the circuit output is correct or not, without knowing the correct output value. It is desirable to design circuits, including checkers, to be self-checking for as many faults as possible. Error detecting codes form the basic framework for the design of a concurrent checking methodology [10]. The use of on-line testing (i.e., checking circuits in conjunction with coded outputs to detect faults) has many advantages over off-line testing, because the output errors are detected immediately upon occurrence and the diagnostic software is eliminated or at least substantially simplified. However, more hardware is required, including the checker, and the additional hardware must be checked too.

Our approach to the on-line testing will be based on the partitioning of the digital system into tiles and using distributed checkers in the tiles to provide the fault location by identifying the checker by which the error is detected [6, 7].

Each circuit tile contains an individual circuit with coded output to be checked, and a checker. The FPGA is partitioned into device tiles equivalent to the circuit tiles, as shown in Figure 2. This operation is performed prior to mapping the circuit onto the FPGA. An optimized algorithm has to be used to reduce the number of checkers in the application circuit to save the area and reduce the total time.

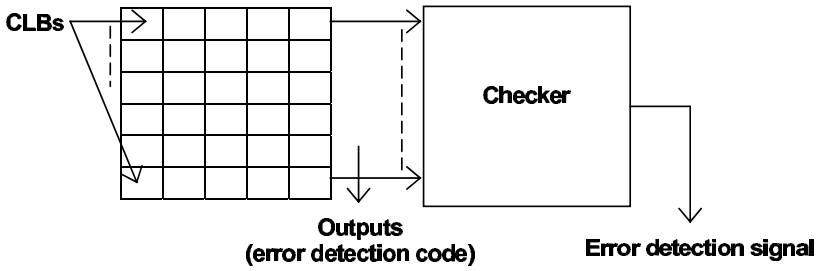

Fig. 2. Proposed FPGA tile

For on-line error detection we chose the Berger code. The structure [1] of a totally self-checking checker for this code is shown in Figure 3. The circuit N_1 generates checks bits from the information bits. The two-rail equality checker, N_2, compares the check bits of the input word with the check bits generated by N_1. Finally, a hardcore logic (exclusive-OR) is used to generate only one error signal.

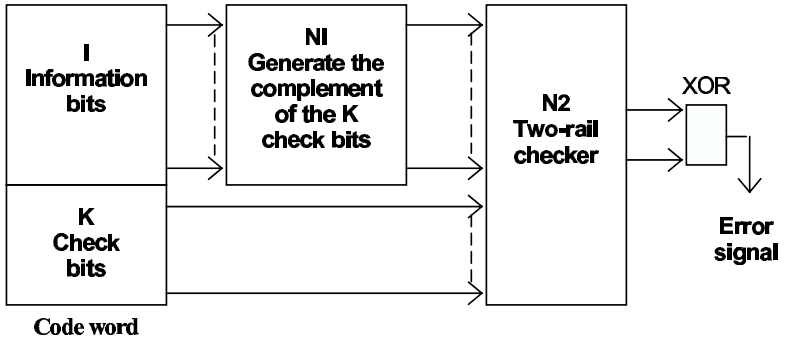

Fig. 3. Totally self-checking checker for Berger code

We evaluated the proposed approach by applying it to six MCNC benchmarks [9]. These benchmarks have the form of two-level AND-OR expressions computed by Berkeley's minimizer Espresso [5]. They do not use coded signals, so the output functions were first encoded in a unidirectional code, such as Berger or m-out-of-n code, which allows the detection of all multiple unidirectional errors.

Each physical circuit was partitioned into different numbers of tiles in order to find the partition that has the minimum overhead by comparing it with the case of the single tile mode, i.e., a single checker for the whole circuit. For example, in the Newcpla2 benchmark, 44 CLBs are used for implementing the circuit in a single tile mode with a maximum delay of 40.6 ns. The number of utilized CLBs for two, three, five, and ten tiles (checkers) is 51, 54, 58, and 58 with a maximum delay of 39.03 ns, 33.84 ns, 40.29 ns, and 35.5 ns, respectively. The best choice of the number of tiles is two if we need a minimum area overhead and three if we need a minimum time overhead.

Table 1 compares the number of CLBs used and the maximum delay in the case of no checker, single checker and distributed checker. This comparison was made using the Xilinx XC4003 family and the Xilinx Foundation CAD tool version '3.1'.

Bench.	#	#	Without checker		Single checker		Distributed checker		
	in	out	CLBs	Delay	CLBs	Delay	CLBs	Delay	Checkers
Dc1	4	7	4	14.2	10	25.31	9	20.38	2
Br2	12	8	63	48	93	57.24	93	62.02	2
Inc	7	9	33	27.4	63	39.05	70	35.9	2
Newtpla2	10	4	15	23.2	24	30.58	22	27.4	2
Squar5	5	8	8	18	15	26.26	15	27.73	2
Newcpla2	7	10	31	28	44	40.6	54	33.84	3

Table 1. Experimental results

The experimental results demonstrate that the area measured by the number of CLBs is nearly duplicated after using the self-checking technique. Comparing single and distributed checkers, the results show that the overhead depends on the application. The maximum area overhead for the distributed tiles approach was 23 % higher than the single tile approach in the Newcpla2 circuit, while the maximum time overhead for the distributed approach was 19 % smaller than the single approach in the Dc1 circuit. The results presented in this paper make the extra hardware required for distributed checkers more attractive in view of its impact on self-test.

5 DIAGNOSIS ALGORITHM

The on-line error detection method proposed in the previous section will locate the fault to within a tile, because all error signals from all tiles are fed into an encoder and the output of the encoder specifies the faulty tile. To locate the faulty CLBs, we must apply an off-line diagnosing algorithm [19], to the faulty tile. As mentioned earlier, we presume that the CLBs may be faulty, while the interconnections are fault-free. The diagnosis algorithm will be applied by rows and by columns and every intersection of the faulty rows and faulty columns indicates a faulty CLB. An external Diagnosis and Reconfiguration Controller (DIRC) unit will control this process.

During fault location, the rows of CLBs of the faulty tile will be configured into pairs. In the first programming phase, the first CLB row will be configured so that every CLB becomes a circuit under test (CUT) and the second row a tree of self-checking checkers — see Figure 4. Here we assume that all CLBs are configured to implement the same logic function in double-rail logic, so that their outputs are always in the 1-out-of-2 codes. In the second programming phase, the rows exchange their roles. The checkers in one row are connected to form a self-checking tree of checkers, while there is no connection between the CUTs. Only one checker is needed to verify the outputs of each CUT. The next checker receives the responses from the previous checker and at the same time receives the outputs from one CUT, and so on. Finally we have a row of CUTs tested by only one row of checkers and the chip I/O blocks (IOBs) directly observe the final responses of the last checker. The same process will be repeated to test the columns, and the intersection of the rows and columns will indicate the faulty CLBs. The CLB logic modules are configured, the test pattern generator is applied off-chip, and each CLB is completely tested.

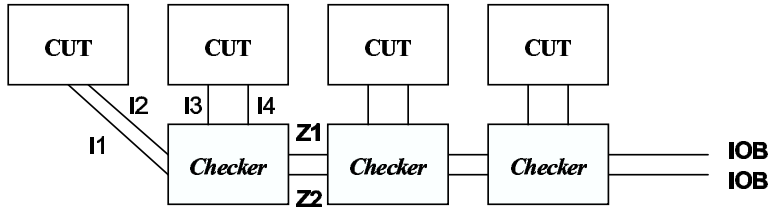

Fig. 4. One programming phase

This diagnosis method can be applied to all types of SRAM-based FPGAs. In this paper we concentrated on Xilinx Virtex device, which can support partial reconfiguration. In the Virtex device, each CLB consists of two slices (S_0, S_1), every slice is equivalent to one CLB cell as in Xilinx XC4000 series. Each slice consists of two LUTs (F, G), multiplexers, two flip-flops, and two unregistered outputs (X, Y). CLB can be configured with configuration memory cells, using the Xilinx Foundation CAD tool, version 3.1. Simulation shows that the application of this approach works well for implementation, i.e., the routing/placement succeeded.

6 CHECKER LOGIC FUNCTION

The logic function implemented by one checker cell should be able to compare outputs of two CUTs, and at the same time to check the outputs of each CUT. The two outputs of each CUT are complemented, for instance configuring the F LUT as XOR and the G LUT as exclusive-NOR (NXOR) [18]. The multiplexer configurations are controlled so that the output of the F LUT will be connected to the X output and the output of the G LUT to Y output. The same test patterns will be applied to all CUTs simultaneously. Therefore their response must be identical, which means that

only the patterns 0101 and 1010 will be accepted by the checker. The first output line of one CUT compares with the first output line of the next CUT, the second output line of the first CUT with the second output line of the next CUT. Similarly, the same checker should be able to check that the two output lines of each CUT are complemented. The results of the comparisons are fed into the next testing cell to compare with another CUT's output lines, etc. Hence, the testing cell can deal with these four inputs and produce two outputs according to Table 2. If both CUT outputs give the same 1-out-of-2 codes, then both CUTs are fault free. In Table 2 we thus do not have to consider the patterns 1001 and 0110, normally accepted by the double-rail checkers. This restriction allows us to use a very simple structure of the checker, as shown in Figure 5.

All CLBs of the faulty tile are fully tested for all single faulty LUTs with two test configurations (TCs) observing two CUT outputs. In [18], only two TCs (XOR, NXOR) are needed to test a LUT of any size. Therefore, in the first TC, F LUTs are configured as XOR and the G LUTs as NXOR. In the second TC, F LUTs are configured as NXOR and the G LUTs as XOR. Two programming phases are used to check the rows of the faulty tile per TC, and other two programming phases for columns per TC. Therefore, the total number of the programming phases needed are eight. At each phase all configured data and test sequences must be loaded from off-chip to the SRAM part and operational inputs of the CLB respectively. With respect to testing a single CLB, we use our fault model to generate the test patterns. A complete test set for one row of CLBs is then easily composed of the patterns generated for an individual CLB.

The test time, which is defined as the time required for the test configurations plus the time required for the test generation [11] is:

$$TestTime \cong 8 \times SRAMsize/frequency, \tag{1}$$

where, frequency is the reconfiguration clock in which the data must be loaded from off-chip. The off-chip memory size for our approach is:

$$OffchipMemorySize \cong 8 \times SRAMsize. \tag{2}$$

I1	I2	I3	I4	Z1	Z2
0	1	0	1	0	1
1	0	1	0	1	0
Any	other	Com	bin.	0	0

Table 2. Truth table of the checker logic function

7 ESTIMATING THE NUMBER OF SPARE CLBS

After locating the faulty CLBs inside the tile, the reconfiguration process has to be started by avoiding the faulty ones and using the spare ones. Therefore, it is

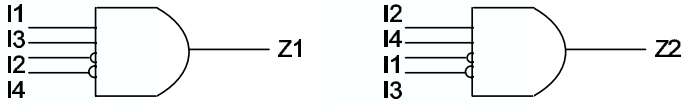

Fig. 5. Checker logic function

necessary to know the number of the spare CLBs before the reconfiguration process takes place. The number of spare cells in a tile of size t utilized CLBs is at least [13, 14]:

$$S_p = \left(\frac{1}{1-p}\right)^{\frac{t}{4}} \tag{3}$$

and at most:

$$S_p = \left(\frac{1}{1-p}\right)^{\frac{t}{2}}, \tag{4}$$

where p is the overall probability of faulty elements distributed in the fashion described in the fault model and t is the number of CLBs.

For instance, with $p = 0.2$, for a circuit tile of size 3×3 CLBs, the parameters t will equal to 9. The number of spare CLBs (Sp), calculated according to (3) is nearly equal to 2. Therefore, the total size of the tile on the device will be 11 CLBs.

Based on (3), or (4), the number of spare CLBs per tile was calculated off line. This number has an effect on the level of FT needed because it specifies the number of faulty elements that can be tolerated inside the tile, the number of all possible different configurations generated per tile, and the storage overhead.

The number of spare elements may be assigned per tile or per subtile. In the case of assigning per tile, the tile will be partitioned into a number of subtiles, each one of the subtiles having at least one spare CLB. For high reliability, the spare elements can be assigned per subtile, where the number of spare elements assigned will be larger and the level of fault tolerance will be high. However, there will be a higher overhead. After assigning the number of spare CLBs per tile and finding that it is smaller than the number of faulty CLBs, the reconfiguration process will only be done if the device still has other unused CLBs.

Table 3 shows the number of spare CLBs in three of the above benchmark circuits, with fixed $p = 0.2$. The number of spare CLBs is first assigned per tile and then the tiles are partitioned into a certain number of subtiles with at least one spare CLB. Secondly, the tiles are first partitioned into a certain number of subtiles and the number of spare CLBs is estimated per subtile depending on its utility CLBs. We note that when the tile size is large, the number of spare elements estimated in each case is nearly the same. However, when the tile size is small, the difference between the numbers of spare elements is larger on the side of the subtile case. This means that the assignment of spares per subtile does not guarantee higher reliability than the assignment of spares per tile, except in the case of small tile sizes.

Bench.	Tile 1			
	Size (CLB)	Sp/tile	Subtiles	
			# Sub-tiles	Sp/Sub-tiles
Newcpla2	21	3.2	5	6.3
Inc	39	8.8	10	12.4
Br2	43	11	11	13.6
Bench.	Tile 2			
	Size (CLB)	Sp/tile	Subtile	
			# Sub-tiles	Sp/Sub-tiles
Newcpla2	16	2.4	4	5
Inc	31	5.6	8	10
Br2	50	16	13	16
Bench.	Tile 2			
	Size (CLB)	Sp/tile	Subtile	
			# Sub-tiles	Sp/Sub-tiles
Newcpla2	17	2.6	4	5
Inc	—	—	—	—
Br2	—	—	—	—

Table 3. The number of spare CLBs

8 PROPOSED FAULT-TOLERANT APPROACH

The faulty tile located by the diagnosis procedure is then circumvented by a new configuration downloaded from an external memory. This is possible because some of the CLBs are used as spares for the rest. Upon detecting an error in a certain tile, DIRC will automatically be used to handle the fault tolerance process. In comparison with the TREC controller used in [2], which supports test, diagnosis, and reconfiguration processes, DIRC will be simpler because the testing process is already done on-line. DIRC will apply a diagnostic algorithm to locate the faulty CLB cells inside the device and then start the reconfiguration process. In the case of devices that support the full reconfiguration, e.g., the Xilinx 4000 family, DIRC will reload the bitstream data to the device, avoiding the faulty CLB cells. DIRC was designed by the JBits software package [8, 16]. In the case of the devices, which support partial reconfiguration, e.g., Virtex family, all possible configurations of the tiles will be generated off-line and stored in an external configuration memory. After locating the faulty element and by using the mapping look-up table, DIRC moves the partial reconfiguration data from the external configuration memory to the adequate positions in the configuration RAM of the FPGA as shown in Figure 6.

9 RECONFIGURATION PROCESS

Our FT approach is based on tiling technique. Each tile is composed of a set of physical resources (CLBs and interconnect), an interface specification that denotes

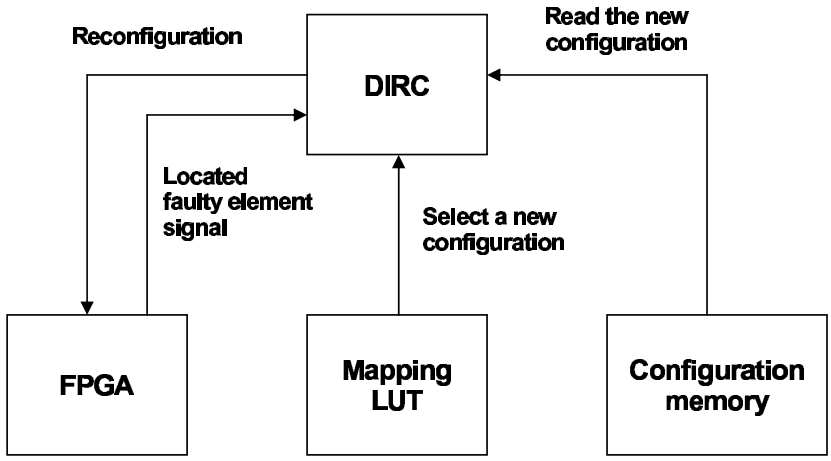

Fig. 6. Structure of the fault-tolerant FPGA system

the connectivity to neighboring tiles, and a net-list. The tiles of small sizes help to reduce the amount of configuration memory by reducing the size of the component that is reconfigured. All possible configurations of the tiles are generated off-line. The number of all possible configurations for a certain tile of area size T CLBs (utility and spare), and tolerating M faulty CLBs are:

$$\begin{pmatrix} \mathrm{r} \\ \mathrm{M} \end{pmatrix}, M \leq S_p.$$

Here we assume that for every set of M faulty CLBs we generate only one configuration.

For example, tile of area size 3×4 CLBs (including three spare CLBs), has 12 possible different configuration files in case of tolerating single faulty CLB, each configuration file has the size of X bytes. Each new configuration is interchangeable with the original, as the interface between the tile and the surrounding areas of the design is fixed and the tile's function remains unchanged. The timing of the circuit may vary, however, due to the changes in routing. All possible configuration bitstream files are indexed into a look-up table. This index will help in selecting the proper configuration file, which does not use the faulty CLBs. For reducing the storage overhead and increasing the reliability, the tile can be partitioned into three subtiles, each subtile consisting of 4 CLBs, including one spare CLB. In this case, only the subtiles which contain the faulty CLBs will be partially reconfigurable, keeping the function performed by the tile. There are 12 possible configurations for the three subtiles, each configuration (of the size about $X/3$ bytes) is independent. Therefore, partitioning the tile into subtiles reduces the storage overhead by about 67 % and allows for tolerating any single fault in a subtile but up to 3 faulty CLBs in the whole tile.

10 DESIGN OF DIRC UNIT USING JBITS

JBits is an application program interface (API) to the Xilinx configuration bitstream [8]. This API permits Java applications to dynamically modify Xilinx Virtex bitstreams. This interface operates either on bitstreams generated by Xilinx design tools, or on bitstreams read back from actual hardware. This provides the capability of designing and modifying circuits for Xilinx Virtex FPGA devices, which is achieved by providing access to all the resources of a Virtex device. The advantage of this approach is primarily that of speed. Java programs performing such reconfigurations can be compiled and run very quickly. JBits may be used as a stand-alone tool or as a basis to produce other tools, including traditional place and route CAD applications, as well as more application specific tools.

The bitstream file of the circuit application can be generated using Xilinx Foundation or using JBits. After detecting an error in a certain tile, the diagnosis algorithm is used to locate the faulty CLBs inside the faulty tile, then the reconfiguration process is started, avoiding the faulty CLBs and using the spare ones as shown in Figure 7. Therefore, it is necessary to know the number of the spare CLBs per tile before the reconfiguration process takes place.

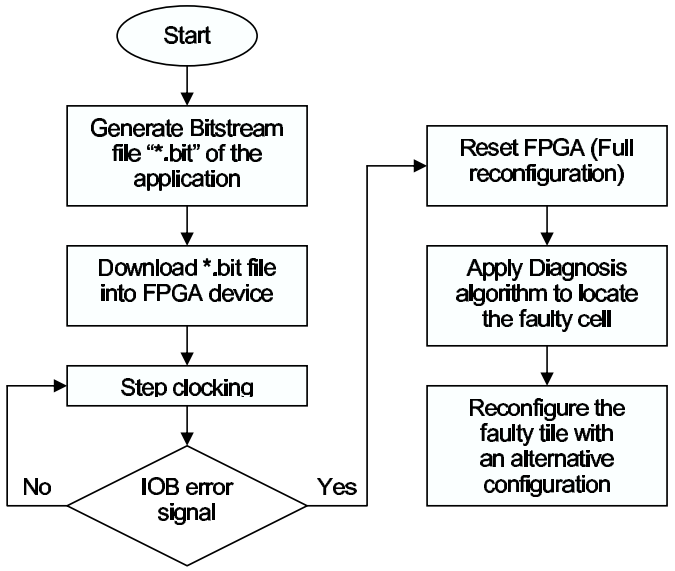

Fig. 7. Shows the high level block diagram for the Jbits code

The DIRC unit performs two main processes for the FPGA: diagnosis of the faulty CLBs and the reconfiguration.

Diagnosis process. After receiving an error signal from a certain tile, and based on the tile size and its CLB addresses inside the FPGA, the DIRC unit will

generate the eight programming phases needed for the diagnosis process and the test pattern generation. The faulty tile will be configured by these programming phases one by one to locate the faulty CLBs by finding the intersection between the error signals generated from its rows and columns as shown in Figure 7.

Reconfiguration process. As mentioned before, all configuration bitstream files for all the tiles are generated offline and placed into a mapping look-up table. These configuration files are indexed to indicate which element(s) they do not use. For instance, tile A of size 2×2 having one spare CLB has four different configuration files. These files are indexed as *A-1*, *A-2*, *A-3*, and *A-4*. Here *A-1* means that this configuration file does not use the CLB number one, and so on. Upon locating the faulty CLBs as a result of applying the diagnosis algorithm, DIRC seeks in the look-up table to find the index of the proper configuration bitstream file, which does not use the faulty CLBs to be downloaded into the same location of the faulty tile. This can be done using the functions listed in Figure 8.

11 CASE STUDY

A DIRC prototype has been successfully implemented based on our technique of reconfiguration using the Virtex Device Simulator (VirtexDS), which simulates at the device/bitstream level, providing an interface, which operates much like actual hardware. All possible bitstream configurations for only one tile of Newcpla2 benchmark of size 3×4 of CLBs including three spare CLBs were generated offline. There are 12 possible bitstream configuration files for this tile for tolerating single faulty CLB, each of size 69.9 kB, and this will result in the storage overhead. We used the BoardScope graphical interface as a debugger — which can access all the resources of the device (LUTs, IOBs, ...) — to the circuits applied onto VirtexDS. We step the clock of the design downloaded onto the FPGA a number of times as a debugging feature using the BoardScope. We debugged the configuration file creating a faulty CLB by modifying its configuration to differ from fault-free CLBs. The error was detected through accessing IOBs by using the Boardscope. Eight diagnosis bitstream files for diagnosing LUTs were generated by the DIRC, and downloaded onto the tile one by one to locate the faulty CLB. The faulty CLB was located and the tile reconfigured by another configuration bitstream file that avoids the faulty CLB. This procedure is currently being transferred to the real Xilinx hardware device.

12 CONCLUSIONS

The reconfiguration method proposed here concentrates on the CLBs, i.e., on the lowest level of the FPGA structure where the reconfiguration is possible. Thus it uses a relatively small area overhead, but the fault location and reconfiguration algorithms become more complex than in case of reconfiguration at a higher level.

```
/* Creating a JBits Object and reading in a bitstream file  */
JBits jbits = new Jbits (Device Type);
Jbits.readPartial (input file);

/* Download the bitstream to device  */
board. SetConfiguration (device, jbits.getPartial ());

/* Reading an error signal from an Input/Output Block (IOB)  */
jbits.getIOB();

/* Generating one of the programming phases for the faulty tile  */
for (row = the row of the first CLB in the tile;
     row< the row of the last CLB of the tile; row++)
for (col = the column of the first CLB in the tile;
     col< the column of the last CLB; col++) {
    int XOR_F[] = Expr.F_LUT ("F1 ^ F2 ^ F3 ^ F4");
    int NXOR_G[] = Expr.G_LUT ("~(G1 ^ G2 ^ G3 ^ G4)");  // the NOT.
        If (XOR_F != null)
        Jbits.set(row, col, LUT.SLICE0_F, XOR_F);
        If (NXOR_G != null)
        Jbits.set(row, col, LUT.SLICE0_G, NXOR_G);
        .
}
/* Generating test patterns */
int [] stimulus = new int []  {0, 1, 0, 1,};
    SetPinValue (pin, stimulus[i]);

/*  Download the chosen partial configuration file
    from the Off-chip memory  */
board.setConfiguration (device, jbits.getPartial());
```

Fig. 8. The Jbits code to perform diagnosis and reconfiguration

A combination of hardware and software tools is used to achieve the FT property of the resulting circuit. Concurrent checkers are used to detect errors during computation, and the faulty CLB is located by a combination of software routine with a set of hardware checkers. The extra hardware required for distributed checkers implies some area and time overhead, however the gain is a good diagnostic resolution. The number of spare elements per tile needed for the reconfiguration process was estimated. Finally, a suitable alternative configuration is selected and downloaded from an external memory in case that the faulty CLB has been successfully located. The advantage of this approach is primarily that of speed. Java programs performing such reconfigurations can be compiled and run very quickly. We have implemented a prototype DIRC system and we demonstrated an application running on VirtexDS with the BoardScope debugging tool. The search for most suitable size and shape

of a tile remains an open problem for further optimization of the fault recovery mechanism.

Acknowledgment

This research was in part supported by grant #102/01/0566 "Built-in Self-Test Equipment Optimization Methods in Integrated Circuits" of the Czech Grant Agency (GACR).

REFERENCES

[1] Abramovici, M.—Breuer, M. A.—Friedman, A. D.: Digital Systems Testing and Testable Design. IEEE Revised Printing, 1990 by AT & T.

[2] Abramovici, M.—Stroud, C.—Hamiliton, C.—Wijesuriya, S.—Verma, V.: Using Roving STARs for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications. Proc. IEEE Intn'l. Test Conf., pp. 973–982, 1999.

[3] Andraka, R. J.: Building a High Performance Bit Serial Processor in an FPGA. Proc. of Design SuperCon '96, 1996, pp. 5.1–5.21.

[4] Antola, A.—Piuri, V.—Sami, M.: On-Line Diagnosis and Reconfiguration of FPGA Systems. Proc. of IEEE Electronic Design, Test, and Application Workshop, Christchurch, New Zealand, 29–31 January 2002, pp. 291–296.

[5] Brayton, R. K.—Hachtel, G.—McMullen, C.—Sangiovanni-Vincentelli, A.: Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publisher, 1984.

[6] Elshafey, K.—Hlavička, J.: An On-Line Self-Checking Approach for Testing FPGA Logic Blocks. 10th International Scientific and Applied Science Conference, ET'2001, Sozopol, Bulgaria, September 26–28, Book 1, pp. 162–167, 2001.

[7] Elshafey, K.—Hlavička, J.: On-Line Detection and Location of Faulty CLBs in FPGA-Based Systems. Proc. of IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS'02), Brno (Czech Republic), pp. 183–190, 17–19. 4. 2002.

[8] `ftp://customer@ftp.xilinx.com/download/JBits2_8.exe`.

[9] `ftp://ic.eecs.berkeley.org`.

[10] Gupta, S. K.—Pradhan, D. K.: Utilization of On-Line (Concurrent) Checkers during Built-In Self-Test and Vice Versa. IEEE Trans., on Computers, Vol. 45, 1996, No. 1.

[11] Haung, W. K.—Lombadi, F.: A General Technique for Testing FPGAs. IEEE VLSI Test Symp., Priceton NJ, pp. 450–455, 1996.

[12] Lach, J.—Mangione-Smith, W. H.—Potkonjak, M.: Low Overhead Fault-Tolerant FPGA System. IEEE Transaction on VLSI System, Vol. 6, 1998, No. 2, pp. 212–221.

[13] Laforge, L. E.: Configuration of Locally Spared Arrays in the Presence of Multiple Fault Types. IEEE Trans., on Computers, Vol. 48, 1999, No. 4, pp. 398–416.

[14] Laforge, L. E.: How to Layout Arrays Spared by Rows and Columns. IEEE International Conference on Innovative Systems in Silicon, 1997.

[15] Metra, C.—Mojoli, G.—Pastore, S.—Salvi, D.—Sechi, G.: Novel Technique for Testing FPGAs. Design Automation and Test in Europe, pp. 89-94, 1998.

[16] McMillan, S.—Guccione, S. A.: Partial Run-Time Reconfiguration Using JRTR. 10th International Conference, FPL 2000, Austria, August 2000, pp. 352–360.

[17] Pradhan, D. K.: Fault-Tolerant Computer System Design. Prentice Hall PTR Publishing Company, 1996.

[18] Renovell, M.: A Specific Test Methodology for Symmetric SRAM-Based FPGAs. FPL 2000, LNCS 1896, pp. 300–311, 2000.

[19] Renovell, M.—Portal, J. M.—Figueras, J.—Zorian, Y.: Testing the Unidimensional Interconnect Architecture of Symmetrical SRAM-Based FPGA. Proc., of IEEE Electronic Design, Test, and Application Workshop, Christchurch, New Zealand, 29–31 January 2002, pp. 297–301.

[20] Xilinx. Inc., `http://www.Xilinx.com`.

Khaled A. M. Elshafey received the BS and MS degrees in system and computer engineering from Al-Azhar University, Cairo, Egypt, in 1991 and 1997, respectively. From 1999-2002, he was awarded a scholarship to be a PhD student at Czech Technical University (CTU) in Prague, Faculty of Electrical Engineering, Department of Computer Science and Engineering, Czech Republic. He received the PhD in 2002. Currently, he works as a teacher in the Faculty of Engineering, System and Computer Engineering Department, Al-Azhar University, Cairo, Egypt.

Jan Hlavička (1942–2002) graduated from Faculty of Electrical Engineering, Czech Technical University in 1964. He received his PhD and DSc degrees from the same university in 1971 and 1987, respectively, and was appointed Associate Professor and Professor by the same institution in 1985 and 1991, respectively. During his fruitful scientific career, he was with the Research Institute for Mathematical Machines in Prague, Siemens Munich, and Department of Computer Science and Engineering of the Faculty of Electrical Engineering, CTU Prague. He was the visiting professor at TH Ilmenau (Germany), Universit de Montréal (Canada) and Hochschule fr Bauwesen Cottbus (Germany). He is the author and co-author of numerous scientific papers. His research interests indluded fault-tolerant computing testing and diagnostics of digirtal circuits an systems,computer architecture, error coding, self-checking circuits.