International Journal of Information Technology and Knowledge Management
January-June 2011, Volume 4, No. 1, pp. 39-42

GENERATING TEST PATTERNS FOR FPGAS USING MULTI-OBJECTIVE GENETIC ALGORITHM

Sangeeta[1] & Vinay Chopra[2]

This paper presents a brief introduction to multi-objective genetic algorithms and FPGAs[4][8]. In this paper we have discussed that how test pattern generation method can be formulated in terms of CNF form [16]and this CNF form can be used to generate test patterns using genetic algorithm[20] We have proposed that by applying a multi-objective genetic algorithm on this CNF form we can increase number of instances to satisfy boolean equation.

Keywords: FPGAs,CNF, Multi-objective Algorithm

1. Multi-objective Optimization

In general, a multi-objective optimization is defined by a function f which maps a vector of decision variables, the so-called decision vector, to a vector of objective values, the so-called objective vector:

$$(y_1, y_2, \ldots, y_n) = f(x_1, x_2, \ldots, x_n)$$

Equation No:1

where $y_1, y_2, \ldots, y_n$ are decision vector variables, $x_1, x_2, \ldots, x_n$ are objective values

Without loss of generality, it is assumed here and in the following that each of the n components of the objective vector is to be maximized[8][10]. In this scenario, a solution (defined by the corresponding decision vector) can be better, worse, equal, but also indifferent to another solution with respect to the objective values "Better" means a solution is not worse in any objective and at least better in one objective than another; the superior solution is also said to dominate the inferior one. Using this concept one can define what an optimal solution is: a solution which is not dominated by any other solution in the search space. Such a solution is called Pareto optimal, and the entire set of optimal trade-offs is called the Pareto-optimal set, which is represented[3]. The concept of Pareto optimality is only the first step in solving a multiobjective optimization problem because at the end a single solution is sought[4][8][10]. Therefore a decision making process is necessary in which preference information is used in order to select an appropriate trade-off. Although there are different ways of integrating this process, in the field of evolutionary multiobjective optimization it is usually assumed that optimization takes places before decision making. That is the goal is to find or approximate the Pareto-optimal set[4].

1,2Department of Computer Science & Engineering Punjab Technical University Jalandhar, India
Email: 1er.virgo@rediffmail.com, 2vinaychopra222@yahoo.co.in

2. Field Programmable Gate Array (FPGA)

Field Programmable Gate Arrays (FPGAs) feature their ability to be configured in the field to implement an arbitrary desired function according to the real-time demands. This ability of FPGAs can help people to achieve a faster design cycle, lower development costs and a reduced time-to market compared to conventional Application-Specific Integrated Circuits (ASICs). FPGAs, therefore, are widely used in many applications such as networking, storage systems, communication, and adaptive computing.

Testing FPGAs requires solutions different from those applicable to ASICs. In literatures, there are several different FPGA testing strategies.

- The first strategy is based on configuring several application circuits and exercising them with test vectors developed specifically for each circuit and supplied by an external tester.
- The second strategy of external testing techniques exploits regular internal structure and reconfigurability of an FPGA to concurrently examine its individual components – configurable logic blocks (CLBs) and interconnects.
- The third strategy of testing techniques for FPGA is based on the concept of Built-In Self-Test (BIST)[13][14]..

2.1. Types of Faults in FPGA

A static RAM based FPGA is composed of a two-dimensional array of configurable logic blocks (CLBs), programmable interconnects, and programmable input/output blocks (IOBs). To realize a specific user-given application on a FPGA chip, a compiler software provided by the FPGA vendor is required to divide the application into several parts with each of them small enough to be fit in a CLB, implement each part in a CLB, and finally connect all used CLBs through a programmed interconnect network[15]. Only if all the programmable resources of the FPGA chip used by the application configuration function correctly, the application can run well on the chip. In order to discuss the application-dependent FPGA testing, it is important to give some detailed information about the fault models widely used and studied in FPGA testing[14][17].

Bridging Fault: The fault represents a short between groups of signals. The logic value of the shorted net may be modeled as 1-dominant (OR bridge), 0-dominant (AND bridge), or indeterminate, depending upon the technology in which the circuit. In FPGAs, bridging faults are the most common failure mode in interconnects.

Stuck-at Fault: The fault is modeled by assigning a fixed (0 or 1) value to a signal line in the circuit and its most popular form are the single stuck-at faults. It is one of the significant failure modes happened in interconnects[2].

Delay Fault: The fault causes the combinational delay of a circuit to exceed clock period. Because the delays caused by interconnection can account for 70% of the FPGA clock cycle period and the programmable interconnects are the primary source of large variations in propagation delays, testing for delay faults in FPGAs should focus on excessive delays in the interconnect network .The C-exhaustive testing (combinationally-exhausive testing) proposed in aims at the detection of delay faults in the interconnect network. Because a programmable FPGA chip, to some extent, can be configured in the field to realize an arbitrary function, FPGA testing can be easily performed through a way of functional testing, the functional defects of the programmable resources in a FPGA chip are also well used in FPGA testing.

3. AUTOMATIC TEST PATTERN GENERATION USING SAT

Boolean Satisfiability (SAT) solvers have been the subject of remarkable improvements since the mid 90s [16].One of the main reasons for these improvements has been the wide range of practical applications of SAT. Indeed, examples of modern applications of SAT range from termination analysis in term-rewrite systems to circuit-level prediction of crosstalk noise. The success of SAT solvers motivated many practical applications, but many practical applications have also provided the examples and the challenges that allowed the development of more efficient SAT solvers. This paper provides an overview of some of the most well-known applications of SAT and outlines several other successful applications of SAT[1][2]. Moreover, the improvements in SAT solvers motivated the development of new algorithms for strategic extensions of SAT. To produce reliable computer systems, defect-free components must be available. Automatic test pattern generation (ATPG) systems distinguish defective components from defect-free components by generating input sets that cause the outputs of a component under test to be different if the component is defective than if it is defect free[2][17].Existing algorithmic ATPG systems for single Stuck-at faults in combinational circuits fall into two classes[6]:

- The structural methods, which perform a topological search of the circuit under test.
- The algebraic methods, which generate test patterns by manipulating algebraic formulas.

The Boolean satisfiability method for test pattern generation is used for single stuck-at faults in combinational circuits that is neither a purely structural method nor an algebraic one. The most successful ATPG systems use structural search methods[14]. Of these, the most notable are.

The D-algorithm, Podem, FAN, and SOCRATES[6]. To generate a test pattern for a single fault, first extract a formula that defines the set of test patterns that detect the fault and then use a Boolean satisfiability algorithm to satisfy the formula.

4. APPLYING GENETIC ALGORITHM TO TEST PATTERN GENERATION

GA can be used in ATPG for exploring the work space. In genetic terms every test vector is considered as a chromosome and set of test vector is called as population. The ATPG performs in 2 phases. To assess every test vector in a population in any generation of evolution. The ATPG algorithm performs in two phases. In the first phase the initial population is being generated with the help of pseudo random process. In the second phase the GA phase the test vectors are evolved based on fitness function[14].The fitness function used is:

```
Fitness = NFi
Where NFi is the number of faults detected.
{
FL= {total number of faults}
initial pop=phase I (FL);
if (FL =NULL)
break;
phase II (initial pop, FL);
}
```

Fig. 1: Pseudo-code of Overall GA Based Test Pattern Generation

Phase I: In this phase the initial sequences composed of M vectors are generated based on pseudo random process. The generated sequences are fault simulated for the faults in the fault list. If the sequence detects fault that fault is removed from the fault list and the corresponding sequence is added into the solution set. If no faults are detected by the sequence, then the last sequence generated in the corresponding cycle is added to the set. This process is repeated for max_iter.

```
Function Phase I
initial pop (FL)
for (i=0; i<max_iter, i++)
{
initial pop=phase I(FL);
randomly generate sequences of length L;
for (each sequence)
{ if sequence detects faults in the fault list
{
add sequence to the test set;
drop the faults detected by that sequence;
}
}
return (initial population);
}
```

Fig. 2: The Pseudo-code of Phase I

```
Function Phase II
{
Initial pop from phase1;
for (l=0;l<no_gen;l++)
{for (k=0; k<popsize;k++)
{select two individuals from
population;
apply crossover with probability 1;
apply mutation with probability 0.01;}
compute fitness of the individuals;
for (each sequence)
if (sequence detects the faults in the fault list
{ add sequence to the solution set;
drop the faults detected by the
sequence;
}
}
}
```

Fig. 3: Pseudo-code of Phase II

Phase II: The initial population of GA is composed of the sequences generated in phase I. To generate a new population from the existing one, two individuals (parents) are selected and crossed to create two entirely individuals (child) and each child is mutated with some small mutation probability. The selection operator is rank based selection. In rank based selection, the solutions are sorted according to their fitness from the worst (rank 1) to the best (rank N).Each member in the sorted list is assigned a fitness equal to the rank of the solution in the list. Thereafter the proportionate selection operator is applied with the ranked fitness value and better solutions are chosen. The two parents are crossed to create two entirely new individuals (i.e.) child and each child is mutated with some small mutation probability. The two new individuals are than placed in the new population and the process continues until the generation is entirely filled. The previous population is discarded. Crossover used is one point crossover. A crossover probability of 1 and mutation probability of 0.01 is used in all circuits. The no_gen is assumed to be 8, to reduce the execution time. During test generation pop_size of 16 is used.

5. Conclusion

SAT problems produce excellent results on various benchmarks. Multi-objective when applied on Boolean SAT makes it more scalable. It can also be applied to solve multiple SAT instances simultaneously in order to increase the speed of execution. Muti-objective optimization is that in which we have to optimize multiple objectives. So this field can be applied to various SAT instances to increase the efficiency of test generation.

References

[1] V. Sivaramalcrishnan Sharad C.Seth, “Parallel Test Pattern Generation Using Boolean Satisfiability”, TH0340-0/0000/00 1991 IEEE.

[2] Tracy Larrabee, Member IEEE 1992 “Test Pattern Generation using Boolean Satisfiability” IEEE Transactions on Computer-Aided Design, 11, No. 1, January 1992.

[3] Carlos M. Fonsecay and Peter J. Flemingz, “An Overview of Evolutionary Algorithms in Multiobjective Optimization”, April 7, 1995.

[4] Paolo Prinetto, Maurizio Rebaudengo, and Matteo Soriza “GATTO: A Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits” IEEE Transactioivs on Computer-aided Design, 15, No. 8, August 1996.

[5] V. Rajesh, Ajai Jain, “Automatic Test Pattern Generation for Sequential Circuits Using Genetic Algorithms” 1063-9667/9 IEEE 1997.

[6] Yong Chang Kim and Kewal K. Saluja, “Sequential Test Generators: Past, Present and Future,” Integration, the VLSI Journal, 26, Issues 1-2,1, Pages 41-54, December 1998.

[7] Carlos A Coello Coello., "A Comparative Survey of Evolutionary based Multiobjective Optimization" December 1998.

[8] Li Shen "Genetic Algorithm Based Test Generation for Sequential Circuits" Institute of Computing Technology, Beijing May 2000.

[9] Ying Gao Lei Shi Pingjin Yao, "Study on Multi-Objective Genetic Algorithm" July 2000.

[10] Arslan, T. Horrocks, D.H. Ozdemir, E. Sch. of Eng., Univ. of Wales Coll. of Cardiff "Structural Synthesis of Cell-based VLSI Circuits Using a Multi-objective Genetic Algorithm," Electronics Letter, 32, Issue 7,651-652 ISSN: 0013-5194,06 August 2002.

[11] Gregor Papa,Tomasz Garbolino ", Franc Novak,Andrzej H lawiczka, "Deterministic Test Pattern Generator Design With Genetic Algorithm Approach Journal of Electrical Engineering", 58, No.3,121–127, 2007.

[12] Charles Stroud, John Sunwoo, Srinivas Garimella, and Jonathan Harris Built-In Self-Test for System-on-Chip: A Case Study0-7803-8580-2/copyright IEEE 2004.

[13] Michael S. Hsiao Virginia Tech, Blacksburg, Virginia VLSI Principles and Architecture, Pages 161-262, 2006.

[14] S.Jayanthy M.C.Bhuvaneswari Sri Ramakrishna Engineering College, Coimbatore, India P.S.G. College of Technology, Coimbatore, India "Simulation Based ATPG for Crosstalk Delay Faults in VLSI Circuits using Genetic Algorithm ICGST-AIML Journal", ISSN: 1687-4846, 9, Issue 2, December 2009.