A Combined Method to Improve Sequential Circuit Test Generation

Roohollah Mohammadkhani and Shaahin Hessabi
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
hessabi@sharif.edu

Abstract

This paper proposes a combined method for sequential circuit test generation which employs STG-based and GA-based test generation techniques. Almost all previous hybrid test generators use algorithmic decomposition, but the proposed method uses circuit decomposition. So the STG-based techniques are used to generate test for control unit and test generation for datapath unit is performed by resorting to GA-based techniques. The combination of the two techniques is expected to provide high fault coverages in a reasonable time. Experimental results show the effectiveness of the approach.

1. Introduction

As the complexity of VLSI circuits grows, test generation for sequential circuits is becoming increasingly more difficult and time consuming. Several different approaches have been proposed in tackling this problem, including time frame-based techniques [1, 2, 3], simulation-based techniques [4, 5] and techniques based on State Transition Graph (STG) [6, 7, 8]. In a typical time frame-based test generator, each fault is first excited and then the fault effects are propagated to a primary output (PO), and the required state is then justified through reverse time processing. A STG-based test generator behavior is similar to time frame-based test generator and the only difference is how to construct the justification and propagation sequences. In a simulation-based test generator, the complexity of backward processing is avoided, and processing occurs in the forward direction only. Candidate test sequences are generated usually by targeting several faults simultaneously; a fault simulator is used to select the best test sequences. Genetic Algorithms (GAs) have been used widely to control the selection of candidate tests.

Comparing these approaches shows that each has its own merits. Hence combination of these methods could be beneficial from the fact that these different techniques are suited for different types of circuits [9].

The proposed test generation method is a combination of GA-based and STG-based techniques. GA-based techniques have been especially effective for data-dominant circuits, and STG-based techniques have also been used widely on control-dominant circuits. This combination which employs GA-based methods, such as adaptive search algorithms in optimization problems that have low execution time, and STG-based methods which have deterministic nature, can provide high structural fault coverage in a reasonable time.

Figure 1 outlines the general view of the suggested strategy. According to this figure, the proposed method is composed of a preprocessing phase and two test generation phases. In the preprocessing phase the STG of the circuit is extracted. After completion of this phase, in the first phase of test generation, the faults located in the datapath unit are considered for test generation and in the second phase the controller faults are targeted.

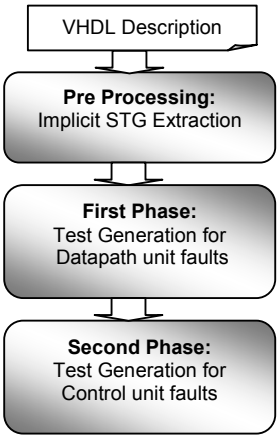

Figure 1. Overall test generation strategy

In these phases, the two mentioned test generation techniques are employed. Therefore, the main focus of the proposed test generation methodology is how these two techniques interact.

Combination of GA-based techniques for datapath test generation and STG-based techniques for the controller provides high fault coverages in an effective manner in terms of test generation time and test set length.

The suggested approach exploits the gate level description of the circuit under test which is decomposed into datapath and control units. The gate level description of a circuit can be obtained through high level or RTL synthesis. In the cases that these two parts are combined, there are some techniques to separate the control and datapath sections of the design [10].

However, for applying the proposed method to any circuit, there is no need for a significant amount of design-for-testability hardware. The hardware overhead of this method is a partial scan chain that includes only those registers that provide status of datapath operations to the control unit. Figure 2 shows the general architecture of the circuits which the proposed test generation method can be applied to.

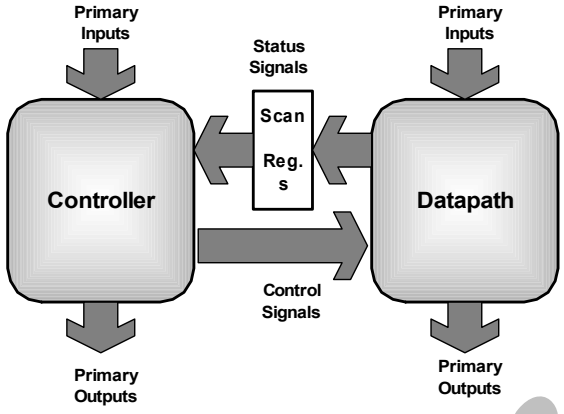

Figure 2. Reference Circuit Model

The remainder of this paper is organized as follows. The overall test generation strategy is presented in Section2. In this section the interaction between the STG-based test generator and the GA-based one is discussed. Section 3 describes how the STG-based test generator works and the application of GA-based test generator is discussed in Section 4. Section 5 presents experimental results and the last section is devoted to conclusions.

2. Overall Test Generation Strategy

The proposed test generation method is composed of a preprocessing phase and two test generation phases. As mentioned previously, the test generation process is entered after completion of the preprocessing phase, in which the STG of the circuit is implicitly extracted. More details of this phase will be presented in the next section. After this preprocessing phase, the first phase of test generation process is started. Pseudo-code of the test generation strategy is shown in Figure 3 According to that, in the first phase of test generation process, faults which are located in the datapath unit are targeted for test generation. After completion of this phase, in the second phase the control unit faults are considered. The main problem in each of these phases is the interaction of two combined test generation techniques, STG-based technique and GA-based technique. In what follows, these two phases will be discussed in more details.

2.1. First Phase

The first phase attempts to generate test sequences that detect faults which are located in the datapath unit. Due to the efficiency of applying the GA-based techniques to data dominant circuits for test generation, in this phase the GA-based technique is employed to generate test sequences. Genetic Algorithms and their application in test generation are described in Section 4. The most important feature of these algorithms which must be considered in this section is that the convergence of a GA in solving a problem is dependent on the precision and size of the initial population. Consequently, in the proposed test generation method, the STG-based method is used to generate test sequences for constructing the initial population.

In these test sequences, only the value of the control unit primary inputs and the value of status signals are specified, so datapath primary inputs are unspecified in these test sequences. The status signals, as outlined in Figure 2, are in the scan chain, so they are fully controllable similar to the primary inputs. The GA, starting from this initial population, attempts to complete the unspecified portion of composed test sequences by targeting the detection of faults in the datapath unit. The generated test sequences are fault simulated and the detected faults are removed from the fault list.

This process is repeated until all detectable faults of datapath unit are removed from the fault list. After completion of this phase, it is expected that all detectable faults of the datapath unit, all faults of the control unit whose effects can be observed only on the controller outputs, and some of the controller faults whose effects have been propagated to the datapath primary outputs are detected. After completion of this phase, the second phase of the test generation process is started.

2.2. Second Phase

In this phase, the remaining undetected faults of the control unit are targeted for test generation. These are faults whose effects in the first phase are propagated to the control signals. In the second phase, these fault effects must be propagated to the datapath primary outputs. Therefore, for any undetected controller faults, a test sequence is generated using the STG-based test generator.

After applying this test sequence to the faulty circuit, the state of the controller signals is saved. This state is given to the PODEM algorithm to propagate the fault effects to datapath primary outputs. If the fault effects cannot be propagated to primary outputs of the datapath unit in one time frame, another time frame is inserted and the PODEM is employed again. If PODEM is successful, the GA is employed in an attempt to justify the state required by PODEM for fault detection.

```
CombinedTG (cirDesc, S0, GAParameters)
{
  //Pre-Processing Phase
    On-OffSets = ExtractSTG (cirDesc, S0);
    faultList = GenerateFaultList (cirDesc)
  //First Phase
  while ( There is improvement in this phase or there are undetected faults in the datapath unit)
  {
        Number_of_test_sequences = 0
        initialPopulation = {}
        do
        {
           targetFault = select_fault (faultList);
           testSequence (Controller PIs)  =  STG-Test (targetFault, On-OffSets, controllerDesc, S0)
           fault_simulate (testSequence);
           Update faultList;
           initialPopulation = initialPopulation + testSequence;
           Number_of_test_sequences ++;
           Update testSet;
        } while (Number_of_test_sequences! = populationSize)
        GA-Test (cirDesc, faultList, initialPopulation, phase1)
   }
   //Second Phase
   while ( There are detectable fault in the controller fault list )
   {
         testSequence (controller PIs) = STG-Test (targetFault, On-OffSets, controllerDesc, S0)
         (prop_Sequence, reuquieredState) = PODEM (Control unit outputs, cirDesc)
          jus_Sequence (datapath PIs) = GA-Test ( reuquieredState, cirDesc, randomPop,
phase2)
          finalTestSequence = merge(testSequence, jus_Sequence) + prop_Sequence
          fault_simulate (finalTestSequence)
          Update testSet;
          Update faultList;
    }
}
```

Figure 3. Pseudo Code of the Proposed Test Generation Strategy

3. STG-based Test Generation

The STG of a sequential circuit, which is the most well known form of reachability information, helps in generating test sequences.

The first feature of STG-based test generation methods is that these methods try to simplify the test generation problem by using the STG of the circuit under test. Therefore, in these methods, the STG of the circuit must be extracted before entering the test generation process. Several different methods have been proposed for extracting the STG of a design [6, 7, 8]. In this paper we use the PODEM-based method which is presented in [7] for STG extraction.

In this method the STG of a design is implicitly extracted using the complete or partial sum-of-product representation of ON/OFF-sets of each flip-flop inputs and primary outputs of the design. ON (OFF)-set of a signal S, is a set of input combinations, which produce a one (zero) value on signal S. After extracting this information, the set of states that are reachable from the reset state and the corresponding transfer sequences for each of them can be computed.

Using the STG in the test generation process (or in other words, knowing the reachable states of the circuit under test) prevents the test generator from justifying invalid excitation states, which significantly reduces the test generation time.

The second feature of STG-based test generation methods [6, 7, 8] is simplifying the test generation problem by dividing it into two or three phases.

In the STG-based test generator which is employed in the proposed approach, the test generation process is composed of three phases which are *error excitation phase*, *justification phase* and *error propagation phase*. The last two phases are performed by using information extracted in the preprocessing phase (STG). These three phases are mentioned below.

1. Error excitation phase: In this phase by applying a combinational test generator such as PODEM, an excitation vector (EV) is found. This vector activates the target fault and propagates its effect to either primary outputs or the next-state lines of the circuit. Upon the application of this vector, the present state of the circuit is called the excitation state.

2. Justification phase: In this phase, a justification sequence (JS) is found to transform the circuit from its initial state R to the excitation state S. This sequence is obtained by traversing the STG of the fault free circuit which is extracted in the preprocessing phase of our suggested method.

3. Error Propagation phase: If under the application of the excitation vector, the fault effect propagates only to one of the next-state lines, a vector sequence must be found which propagates the error effect to one of the primary outputs. This sequence, called differentiating sequence (DS), is also constructed by utilizing STG of the fault-free circuit.

Sequences which are obtained from STG of a fault free circuit may not be valid in the faulty condition, so after phase three the entire test sequence composed of JS, EV and DS is fault simulated to see whether the fault under test or any other undetected faults are detected. If the test is not valid under faulty condition and the maximum number of test sequences is not reached, another fault will be selected for STG-based test generation.

4. GA-based Test Generation

GAs are the search and optimization algorithms which tend to converge on solutions that are globally optimal or nearly so [11, 12]. GAs are used in both of the two phases of the proposed test generation method. We use a simple GA in both of these phases. In the first phase, a GA is employed that contains a population of individuals or strings. Each individual in this phase is a sequence of test vectors and has an associated "fitness" which measures its quality in terms of detected faults. The initial population is composed of N partially-specified test sequences which are provided by the STG-based test generator. As mentioned before, constructing the initial population from the test sequences generated by the STG-based test generator enhances the efficiency of the GA and helps it generate valid test sequences. In other words, by starting from these partially-specified test sequences, the GA attempts to generate fully specified test sequences by targeting all datapath faults. The specified parts of these test sequences are the values of control signals, and the unspecified sections are the values of the datapath inputs. According to the experiences gathered in test generation, in most circuits the test sequences generated for stuck-at faults, especially in the datapath section, are usually clustered instead of being distributed. So the test sequences generated this way have a good chance of detecting new faults.

After constructing the initial population, the evolutionary processes of *selection*, *crossover* and *mutation* are used to generate an entirely new population from the existing population. Not all of the test sequences of the newly generated population are added to the final test set. Only the test sequences which detect new faults are added to the test set to keep the test set compact. The process of generating a new population from an existing one, repeats until acceptable fault coverage for the datapath unit faults is achieved or the number of generation is reached an upper limit.

In the second phase, as mentioned before, the GA is used to propagate the effect of the controller faults, from the control signals to the primary outputs of the datapath unit. The functionality of the GA in this phase is very similar to the one used in the first phase. The main difference is the definition of Fitness function. The quality of an individual or its associated fitness measures the number of required flip-flop values that are correctly justified by each candidate sequence (individual).

The simple GA which is employed in two phases utilizes adopted the *tournament selection*, *uniform crossover* and *random mutation* operators. The selection operator selects one or more of the fittest individuals in a population to apply mutation or crossover operators. The crossover operator mates two different test sequences to generate a new one and mutation is done by simply flipping a randomly selected bit of a test vector in a sequence (an individual).

5. Experimental results

The STG-based test generator (STG-Test) is implemented in 5,000 lines of C++ code; it uses the PODEM algorithm (which is implemented in C++) to implicitly extract the STG of the circuit under test. The GA-based test generator (GA-Test) is also implemented in 2,000 lines of C++ code. The STG-Test and GA-Test programs, according to pseudo code of Figure 3, are invoked in the CombinedTG program to implement the proposed test generation methodology.

To verify our approach, we used a simple processor, SAYEH. We applied our test generation method to this processor. Figure 4 shows the architecture of this processor. The processor has a 16-bit data bus and a 16-bit address bus and 8- and 16-bit instructions. Short instructions may contain shadow instructions, which effectively pack two such instructions into a 16-bit word. The controller of SAYEH has five states*: reset, halt, fetch, decode,* and *exec*. The required scan-chain for circuit decomposition in the proposed method includes 16 flip-flops for SAYEH processor.

To evaluate the effectiveness of the proposed method, we compared its fault coverage, number of deterministic ATPG calls and final test set length with a purely GA-based test generator, and again with a deterministic test generator.

The values used as Genetic Algorithm parameters, such as population size, mutation and crossover rate, tournament size, overselection rate and the maximum number of generations, are reported in Table 1. These parameters are tuned to improve the effectiveness of the GA used in the proposed method for test generation.

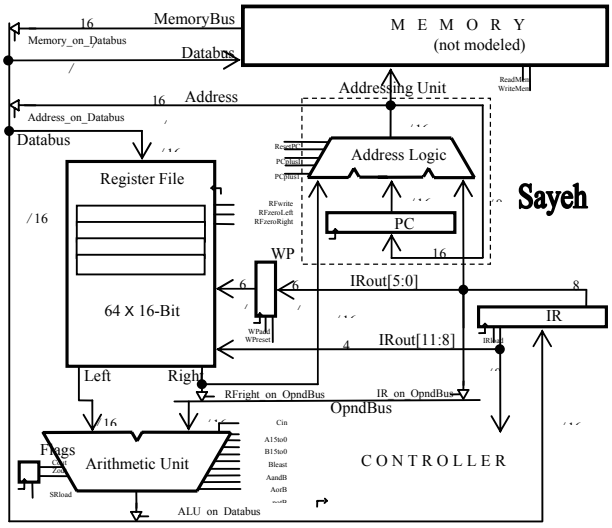

Figure 4. Architecture of SAYEH processor

Table 1. Genetic Algorithm parameters

Parameter	Value
Population size	10
Mutation rate	0.1
Crossover rate	1
Tournament size	4
Overselection rate	0

The results we obtained are presented in Table 2. These results show that the proposed method has higher fault coverage compared to the GA-based test generator. Moreover, the number of deterministic ATPG calls in the proposed method, which have direct influence on the test generation time, is much less than the pure deterministic test generation.

Table 2. Comparison of results for three test generation methods

Test generation method	Fault coverage	Number of Deterministic ATPG call	Test set Length
GA-based	81.6%	0	2560
Deterministic	95%	110	1380
Proposed Method	95%	73	1635

6. Conclusions

In this paper, an approach for sequential circuit test generation is presented. The proposed method is a combination of STG-based and GA-based methods, so this combined approach benefits from the best features of these two test generation techniques. STG-based methods have good performance in test generation for control-dominant circuits because they use the reachability information of the circuit under test and this information can be easily extracted for controllers. GAs are the search and optimization algorithm that have fast execution time because of their non-deterministic nature. It has been shown that the GA-based test generation methods are most effective for data-dominant circuits. Therefore, in the proposed test generation approach the STG-based method is employed for the control unit test generation and a GA-based method is used for datapath unit test generation. The fast execution run of the GA-based test generator, combined with a powerful and deterministic STG-based test generator that can identify undetectable faults, provides high fault coverage in a reasonable time. Experimental results gathered on a simple processor show the feasibility and effectiveness of the method in terms of achieved structural fault coverage, number of deterministic ATPG calls and final test set length.

References

[1] T. M. Niermann, J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," In Proceedings of the European Design Automation Conference, Feb. 1991, pp. 214 –218.

[2] T. P. Kesley, K. K. Saluja, S. Y. Lee, "An Efficient Algorithm for Sequential Test Generation," IEEE Trans. on Computers, Nov. 1993, pp. 1361-1371.

[3] I. Hamzaoglu, J. H. Patel, "Deterministic Test Pattern Generation Techniques for Sequential Circuits," ICCAD 2000, pp. 538-543.

[4] R. Guo, S. M. Reddy, I. Pomeranz, "On Improving a Fault Simulation Based Test Generator for Synchronous Sequential Circuits," Asian Test Symposium, 2001.

[5] D. G. Saab, Y. G. Saab J. A. Abraham, "CRIS: A Test Cultivation Program for Sequential VLSI Circuits," In Proceedings of the IEEE International Conference on Computer-Aided Design, Nov. 1992, pp. 216–219.

[6] H.-K. T. Ma, S. Devadas, and A. Sangiovanni-Vincentelli,"Test Generation for Sequential Circuits," IEEE Trans. Computer-Aided Design, Vol.7, pp.1081-1093, Oct. 1988.

[7] A. Ghosh, S. Devadas, A. R. Newton, "Test Generation and Verification for Highly Sequential Circuits," IEEE Transaction Computer Aided Design, May 1991, pp. 652– 667.

[8] H. Cho, G. D. Hatchel, F. Somenzi, "Redundancy Identification/Removal and Test Generation for Sequential Circuits Using Implicit State Enumeration," IEEE Trans. Computer-Aided Design, vol. 12, July 1993, pp. 935-945.

[9] E. M. Rudnick, J. H. Patel, “Combining Deterministic and Genetic Approaches for Sequential Circuit Test Generation,” In Proceedings of the ACM/IEEE 32nd Design Automation Conference, June 1995, pp. 183–188.

[10] D. Corvino, I. Epicoco, f. Ferrandi, F. Fummi, and D. Sciuto. “Automatic VHDL reconstructing for RTL synthesis optimization and testability improvement,” Proc. IEEE Int. Conf. Computer Design, 1998, pp. 587-596.

[11] M. S. Hsiao, “Sequential Test Generation Using Genetic Methods,” PhD Thesis, Illinois University, 1997.

[12] X. Yu, A. Fin, F. Fummi, E. M. Rudnick, “A Genetic Testing Framework for Digital Integrated Circuits,” ICTAI 2002, pp. 521-526.