NAIST-IS-DD0561023

Doctoral Dissertation

Studies on Test Generation and Design for Testability Based on Knowledge for LSIs

Masato Nakazato

March 23, 2007

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Masato Nakazato

Thesis Committee:

Professor Hideo Fujiwara	(Supervisor)
Professor Hiroyuki Seki	(Co-supervisor)
Associate Professor Michiko Inoue	(Co-supervisor)

Studies on Test Generation and Design for Testability Based on Knowledge for LSIs*

Masato Nakazato

Abstract

LSI (Large Scale Integration) circuits are basic components of today's complex digital systems. As LSI circuits with high performance and a lot of functions are produced by very deep submicron manufacturing (VDSM) technologies, the LSI circuits have many problems for the test. As density of an LSI circuit grows beyond billions of gates, the complexity of a test generation for the LSI circuit is increasing. Moreover, the produced transistors have non-uniform characteristic and size caused by hurdles imposed by the fundamental laws of the nano-electronics physics. These cases will cause long test generation time and faults related to timing of LSI circuits. Therefore, LSI testing will be required to generate a test in short time and perform at-speed testing.

In this dissertation, in order to satisfy these requirements, we propose a test generation method for sequential circuits and a design for testability (DFT) method for processors based on knowledge, which eases the testing, extracted from the high level design of LSIs. The proposed test generation method consists of a synthesis for testability (SFT) method and a test generation method using knowledge extracted from the proposed SFT. The proposed test generation method can achieve 100% fault efficiency in short time and enables at-speed testing. We also propose special DFT for software-based self-test (SBST) which generates a test in reasonable time for a processor. However, the SBST have a problem of "error masking" where errors disappear into a circuit. For any test program generated by the SBST, the proposed DFT method can completely resolve error masking by adding observation points to the original design, and enables at-speed testing.

Keywords:

sequential test generation, software-based self-test, design for testability, test knowledge, at-speed testing

* Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0561023, March 23, 2007.

List of Publications

Journal Paper

1. Masato Nakazato, Satoshi Ohtake, Kewal K. Saluja and Hideo Fujiwara, "Acceleration of Test Generation for Sequential Circuits Using Knowledge Obtained from Synthesis for Testability," *The IEICE Transactions on Information and Systems*, Vol.E90-D, No.1, pp.296-305, Jan. 2007.

International Conferences (Reviewed)

1. Masato Nakazato, Satoshi Ohtake, Kewal K. Saluja and Hideo Fujiwara, "Acceleration of Test Generation for Sequential Circuits Using Knowledge Obtained from Synthesis for Testability," *Proceedings of the 6th IEEE Workshop on RTL and High Level Testing (WRTLT'05)*, pp.50-60, Jul. 2005.

2. Masato Nakazato, Satoshi Ohtake, Michiko Inoue and Hideo Fujiwara, "Design for testability of software-based self-test for processors," *Proceedings of the 15th IEEE Asian Test Symposium (ATS'06)*, pp.375-380, Nov. 2006.

3. Ilia Polian, Bernd Becker, Masato Nakazato, Satoshi Ohtake and Hideo Fujiwara, "Period of grace: a new paradigm for efficient soft error hardening," *18. ITG/GI/GMM Workshop Testmethoden und Zuverlassigkeit von Schaltungen und Systemen*, pp.41-45, Mar. 2006.

4. Ilia Polian, Bernd Becker, Masato Nakazato, Satoshi Ohtake and Hideo Fujiwara, "Low-cost hardening of image processing applications against soft

errors," *Proceedings of the 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'06)*, pp.274-279, Oct. 2006.

Technical Reports

1. Masato Nakazato, Satoshi Ohtake, Kewal K. Saluja and Hideo Fujiwara, "Acceleration of Test Generation for Sequential Circuits Using Knowledge Obtained from Synthesis for Testability," *Technical Report of IEICE (DC2004-97)*, Vol. 104, No. 664, pp.33-38, Feb. 2005. (In Japanese)

2. Masato Nakazato, Satoshi Ohtake, Michiko Inoue and Hideo Fujiwara, "Design for Testability of Software-Based Self-Test for Processors," *Technical Report of IEICE (ICD2006-40)*, Vol. 106, No. 92, pp.49-54, Jun. 2006. (In Japanese)

3. Nobuhiro Yamagata, Masato Nakazato, Kazuko Kambe, Tomokazu Yoneda, Satoshi Ohtake, Michiko Inoue and Hideo Fujiwara, "DFT of Instruction-Based Self-Test for Non-pipelined Processors," *Technical Report of IEICE (DC2005-73)*, Vol. 105, No. 607, pp.7-12, Feb. 2006. (In Japanese)

Awards

1. IEEE Kansai Section Student Paper Award, Feb. 2006.

2. The 6th IEEE Workshop on RTL and High Level Testing (WRTLT'05) Best Paper Award, Nov. 2006.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, digital systems are widely used in various aspects of daily life. The key components of digital systems are LSI (Large Scale Integration) circuits, and malfunctions of the circuits will affect the behavior of digital systems. The incorrect behavior of digital systems causes serious accidents if the digital systems are used as lifeline systems. The expectation of zero failure can only be met if all manufacturing defects are eliminated. A key requirement for obtaining reliable electronic systems is the ability to determine that systems are error-free. LSI testing plays an important role in satisfying this requirement. LSI testing is to check whether faults exist in a circuit, and it consists of two main phases; test generation and test application. In test generation, a test sequence that is an input sequence to detect faults is generated. In test application, the generated test sequence is applied to the circuit.

The LSI technology has several complexities such that billions of transistors are put on a single chip and the chip is implemented with GHz clock frequency and so on. If the physical gate length is shorter, it is effective in implementing these things. The physical gate length of today's semiconductor process technology is 35nm or 50nm. This technology is utilized for producing System-On-a-Chip (SOC) so that processors, digital signal processors, memories and some modules consisting of sequential circuits etc. are integrated on a silicon wafer, state-of-the-art processors (e.g. Athlon64-X2, Opteron by AMD Core and Core2 by Intel).

According to the prediction of the 2005 International Technology Roadmap for Semiconductors (ITRS) in Figure 1.1, the physical gate length of ASIC (SOC

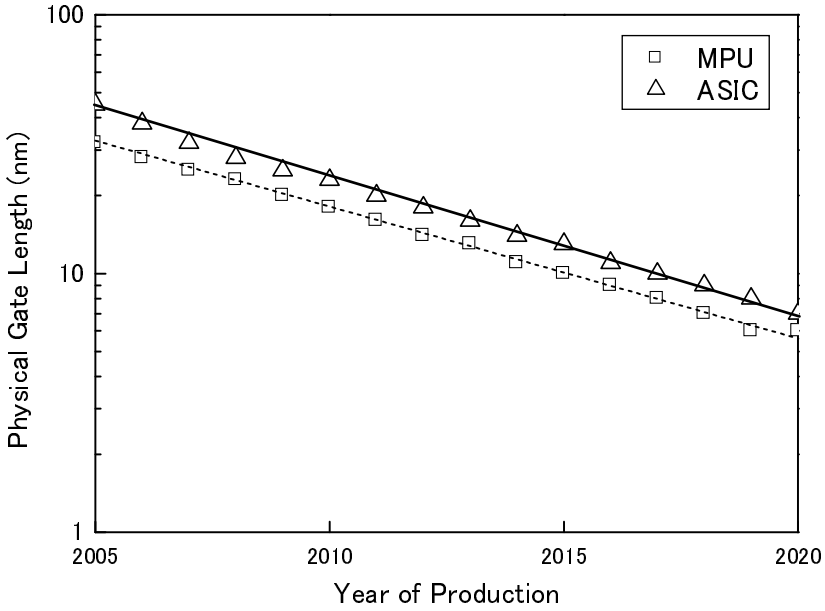


Figure 1.1. 2005 ITRS Product Technology Trends

is one of ASIC) will drop 7nm and the physical gate length of processors will drop 6nm in 2020. A few years ago, the implementation of correctly operating electronic circuits in such a small geometry - usually referred as Very Deep Submicron Manufacturing (VDSM) technologies - was believed to be extremely difficult if at all possible. Because they have hurdles imposed by the fundamental laws of the nano-electronics physics when circuit elements are manufactured. However, VDSM technologies are successfully used today to produce high performance circuits.

As LSI circuits with high performance and a lot of functions are produced by VDSM technologies, the LSI circuits have many problems for the test. As density of an LSI circuit grows beyond billions of gates, the complexity of a test generation for the LSI circuit is increasing. Moreover, the produced transistors have non-uniform characteristic and size caused by hurdles imposed by the fundamental

laws of the nano-electronics physics. The length between signal lines in the LSI circuit is also very short. Two signal lines are easily shorted by dust that gets mixed during producing LSIs and a signal line may have high resistance. These cases will cause long test generation time and faults related to timing of LSI circuits. Therefore, LSI testing will be required to generate a test in short time and perform at-speed testing.

In this dissertation, in order to satisfy these requirements, we propose a test generation method for sequential circuits and a design for testability (DFT) method for processors based on knowledge, which eases the testing, extracted from the high level design of LSIs. The proposed test generation method consists of a synthesis for testability (SFT) method and a test generation method using knowledge extracted from the proposed SFT. The proposed test generation method can achieve 100% fault efficiency in short time and enables at-speed testing. We also propose special DFT for software-based self-test (SBST) which generates a test in reasonable time for a processor and enables at-speed testing. Although SBST has many advantages, in experiments of [11-17], the high fault efficiency is not achieved for processors. The reason why faults can not be detected is "error masking" where multiple errors mask each other, and any error is not propagated to any primay output. For any test program generated by the SBST, the proposed DFT method can completely resolve error masking by adding observation points to the original design, and enables at-speed testing.

The rest of this dissertation is organized as follows. Chapter 2 gives the basics of test method for sequential circuits and processors, and design for testability method which is generally utilized for easing LSI testing. In Chapter 3, we propose a test generation method for sequential circuits based on knowledge obtained from synthesis for testabiliy. The sequential circuit is synthesized from a given FSM by a synthesis for testability (SFT) method proposed in this chapter which takes the features of our test generation method into consideration. The SFT method guarantees the existence of state distinguishing sequences of the specified length by making the given FSM reduced. Thus, the performance of the test generator is improved as it uses state justification sequences extracted from the completely specified state transition function of the FSM produced by the synthesizer. The proposed method can completely identify every fault in the circuit

obtained by the proposed SFT method to be detectable or untestable. In our experiments, 100% fault efficiency is achieved for all the benchmark circuits in relatively short test generation time. Chapter 4 proposes design for testability method which completely resolves the problem of error masking for any test programs generated by the template-based software-based self-test approach. The proposed method adds only observation points to the original design, and it enables at-speed testing and does not induce delay overhead. Finally, in Chapter 5, we conclude these works and discuss directions for future work.

Chapter 2

Test Generation and Design for Testability

1. Fault Model

The consideration of possible faults in a digital circuit is undertaken in order to establish a minimum set of test vectors, which collectively will test whether faults are present or not. If none of the predefined faults are detected, then circuit is considered to be fault-free. There are several fault models presented in the literature to model various defects. This section presents widely used fault model which is related to my study; namely stuck-at fault model, which deal with a logic.

A stuck-at fault is assumed to affect only the interconnection between the gates. Each connecting line can have two types of faults: stuck-at-0 (*s-a-0*) and stuck-at-1 (*s-a-1*). Thus, a line with s-a-0 fault is fixed to have a value 0.

In general, several stuck-at faults can be simultaneously present in the circuit. A circuit with n lines can have $3^n - 1$ possible stuck line combinations, because each line can be in one of three states: *s-a-0*, *s-a-1*, or fault-free. All combinations except one having all lines in fault-free states are counted as faults. Clearly, even a moderate value of n will give an enormously large number of multiple stuck-at faults. It is common practice, therefore, to model only single stuck-at faults. An n-line circuit can have at most $2n$ single stuck-at faults. This number is further reduced by fault collapsing technique.

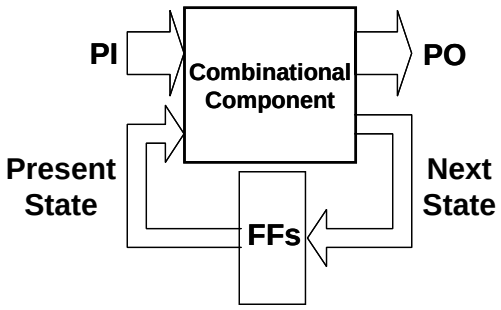


Figure 2.1. A sequential circuit

2. Test Generation of Stuck-at Faults for Sequential Circuits

This section describes sequential test generation and the previous work. Both combinational test generation and sequential test generation are NP-complete problem. For combinational test generation, there are a lot of results of studies. These test generation methods generate a test for a large combinational circuit in relatively reasonable test generation time. On the other hand, sequential test generation is difficult to generate a test in relatively reasonable test generation time compared with combinational test generation because sequential circuits have combinational circuit's parts and flip-flops. (Figure 2.1)

Most sequential test generation methods generally utilize a time frame expansion model. A time frame is the combinational circuit extracted from a sequential circuit by treating its present state lines and next state lines as pseudo primary inputs and pseudo primary outputs, respectively.

In Figure 2.2, a time frame expansion model for a sequential circuit is a combinational circuit constructed by connecting time frames such that the pseudo primary outputs of a time frame t is connected to the pseudo primary inputs of a time frame $t + 1$. Sequential test generation problem is reduced to combinational test generation problem by utilizing the time frame expansion model. However, the time frame expansion model of the sequential circuit has multiple faults because every time frame has the same stuck-at fault. We must consider

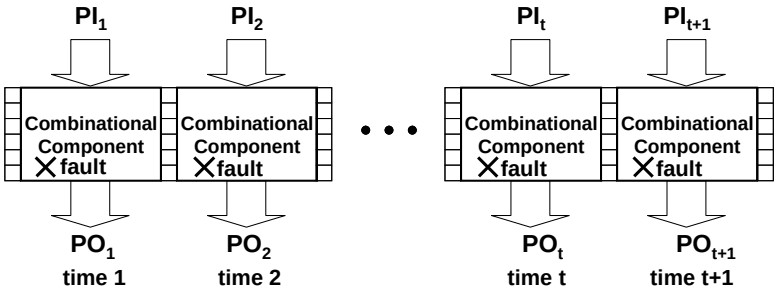


Figure 2.2. A time frame expansion model

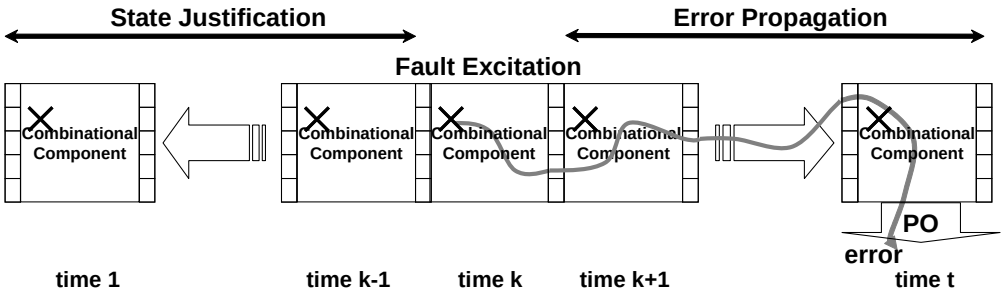


Figure 2.3. General sequential test generation

combinational test generation for the combinational circuit which has multiple faults.

Figure 2.3 shows general sequential test generation method. The sequential test generation method consists of three processes: fault excitation, state justification and error propagation.

At first, for a target fault, fault excitation finds an excitation vector which is assigned to primary inputs and pseudo primary inputs to produce errors and to propagate them to the primary outputs and/or the pseudo primary outputs of the fault excitation frame. The pseudo primary input part of an excitation vector is referred to as an excitation state.

Once an excitation vector is found, state justification is performed. In order to justify from the initial state to the excitation state, state justification process

searchs values assigning to primary inputs. A lot of backtracks occur during performing this process in order to transfer the initial state to the excitation state.

If the fault is not identified as detected or untestable by the first two processes, error propagation is performed. The error propagation process determines primary input values of the expanded time frames to propagate an error to a primary output. A lot of backtracks also occur during performing this process in order to propagate an error.

There are a lot of results of studies for sequential test generation algorithm. This section describes sequential test generation algorithms which are related to my study: FASTEST, HITEC and VERITAS.

The FASTEST[6] is incomplete sequential test generation method. This method can not identify untestable faults. Our objective is that the test generator can completely identify a fault as detectable or untestable.

The HITEC[2] is a well known test generator for sequential circuits. This method consists of two phases. The first phase is the forward time processing phase in which a fault is excited and the resulting fault effect is propagated to a primary output. The second phase is the backward time processing phase which justifies the state required for activating the fault. Then, this method is not efficient to generate tests because it still backtracks during the execution of each phase.

The VERITAS[3] test generation method is an extension of the finite state machine (FSM) verification approach. This method constructs a product machine of a good FSM and its faulty version, and carries out reachability analysis by traversing the product machine. The information obtained by the reachability analysis is used to generate a test sequence. Although this simplifies generation of state justification sequences, it is not efficient to generate tests because it has to deal with huge product machines.

Therefore, these test generation methods are not efficient about test generation time.

3. Software-Based Self-Test

This section describes Software-Based Self-Test (SBST) approach and the previous works of software-based self-test for the testing of processors.

SBST links functional testing with gate level fault model. The concept of the SBST is illustrated in Figure 2.4. It uses on chip resources and processor instructions to deliver the test patterns and collection of test responses. Self-test routines are stored in instruction memory area and data which they need for execution are stored in data memory area. Both transfers (instructions and data) are performed using external test equipment. The external test equipment that transfers test data at high speed and has a large memory is not required. Tests are applied to modules of the processor during the execution of the self-test programs and test responses are stored back in the data memory area.

At first, the self-testing code is downloaded to the processor instruction memory area of the processor via external tester which has access to the internal system bus. The self-test data is downloaded to the data memory area of the processor via the same external tester. Self-test data may consist of parameters, variables called by the execution of the self-test code.

Once self-test code and data are transferred to processor memory, the control is transferred to self-test program which starts execution of self-test. Test patterns are applied to internal processor module via processor instructions to detect their faults. Test responses of the applied test instructions are collected in registers and/or data memory area.

After self-test code completes execution, the test responses are transferred to the external tester in order to evaluating the expected fault-free test responses.

Application of test patterns to processor via processor instructions consists of the following three steps, which are shown in Figure 2.5.

Justification of test patterns: Test patterns is transferred to a module under test in a processor by the execution of a test program. This step may require one or more processor instructions.

Application of test patterns and response collection: Test patterns are applied to the processor's module under test and module's response are

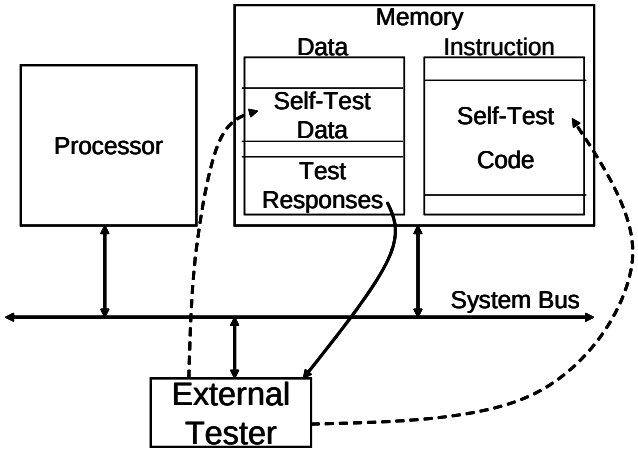


Figure 2.4. Software-Based Self-Test

collected in registers or memory. This step may require a few processor instructions.

Observation of response: Response collected internally are exported towards data memory. This step may also require one or more processor instructions.

For example, a fault inside a forwarding control unit of Dlx_N processor referred to Appendix A can be tested by the instruction sequence in Figure 2.6.

Instructions from I1 to I8 correspond to justification of test patterns as shown in Figure 2.5(a). Instructions from I9 to I11 correspond to application of test patterns and test response collection as shown in Figure 2.5(b). Instructions from I12 to I14 correspond to observation of internal test response in Figure 2.5(c). Instructions from I9 to I11 transfer test patterns to adjacent registers of the forwarding control unit. The value which is calculated by the execution of instructions from I9 to I11 is the test response of the module under test. Therefore, Instructions from I1 to I8 transfer values which are required to execute instructions from I9 to I11. When the processor executes instruction I9, the processor requires loading the value from the data memory. Therefore, the required values are already set to the data memory before executing a test program.

Finally, Instructions from I15 to I17 transfer the test response to the data

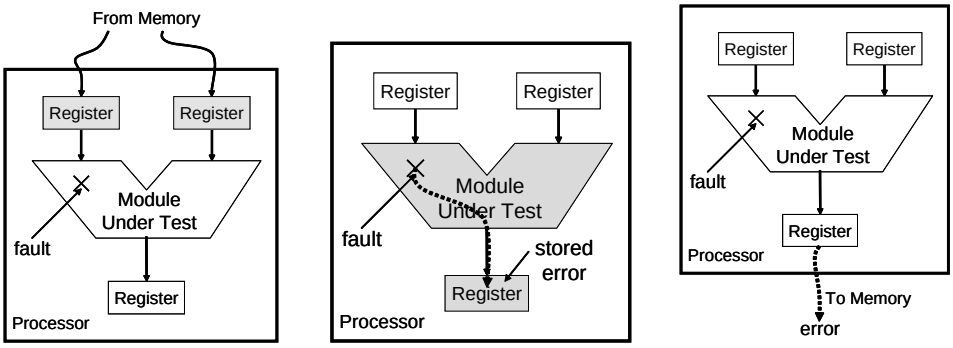


Figure 2.5. Application of software-based self-test: (a) Justification of test patterns; (b) Application of test patterns and response collection; (c) Observation of response.

memory.

Due to inherent non-intrusive approach, SBST has following advantages:

No area overhead: This approach uses only processor resources for test application and test response collection. Therefore, it does not lead to area overhead.

No performance degradation: This approach does not modify the circuit under test. Therefore, it does not lead to performance degradation.

At-speed test: This approach always applies test vectors at-speed as it uses functional mode of operation. Therefore, it can be easily used for the testing of timing faults.

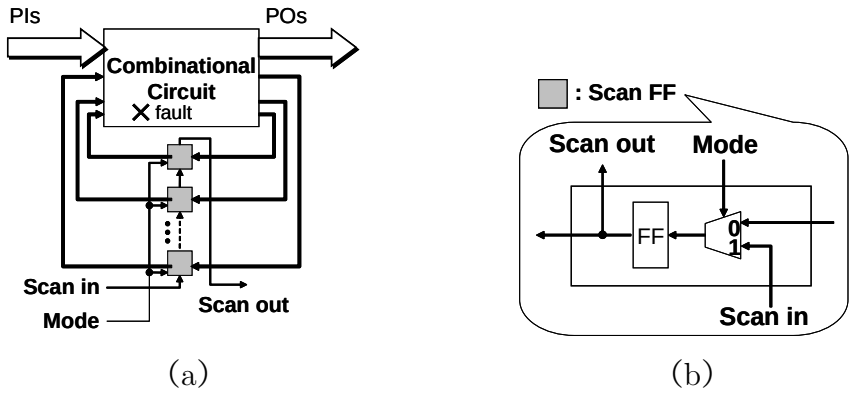
Due to the above advantages of SBST, it is a suitable testing methodology for processor testing. In the next, we describes the previous work of software-based self-test of processors.

Some methods among the SBST methods generate a test program based on test program templates targeting structural faults to achieve the high fault coverage [13-17]. In this approach, gate-level test generation is applied for each module under test (MUT) of a processor (MUT test generation), and a test program is synthesized from a test pattern generated in MUT test generation (test program

```
I1:   LHI     r1, X"0000"
I2:   ADD.I   r2, r1, X"1111"
I3:   LHI     r3, X"1111"
I4:   ADD.I   r4, r3, X"0000"
I5:   LHI     r5, X"1010"
I6:   ADD.I   r6, r5, "0101"
I7:   LHI     r7, X"1001"
I8:   ADD.I   r8, r7, X"0110"
I9:   LW      r7, r4(X"0000")
I10:  ADD     r11, r6, r8
I11:  SUB     r12, r8, r2
I12:  LHI     r13, X"1000"
I13:  ADD.I   r14, r13, X"0001"
I14:  SW      r12, r14(X"0000")
```

Figure 2.6. The instruction sequences for testing a forwarding control unit of Dlx_N processor.

synthesis), where a test program justifies the test pattern from the memory to the MUT and propagates the test response from the MUT to the memory. To guarantee the test program synthesis, test program templates are used. A test program template is an instruction sequence with unspecified operands that delivers test patterns to an MUT and observes the test responses. The approach extracts constraints from each template since the template represents ways to propagate tests from the memory and test responses to the memory, and applies test generation for the MUT under such constrains. In this approach, we can easily synthesize a test program from a test pattern for the MUT. However, the justification and observation parts consider only behavior of a fault-free processor and do not consider behavior of a faulty processor, and such parts do not work as expected. In this case, some faults detected by a test pattern for a MUT may not be detected by the synthesized test program. We call such a phenomenon "error masking."



(a) (b)

Figure 2.7. The full-scan design method: (a) A sequential circuit designed by the full-scan design method; (b) A scan-flip-flop (SFF).

4. Full-Scan Design

This section describes full-scan design method[1] which is one of DFT methods. This method is widely utilized for easing LSI testing.

Figure 2.7 (a) shows the full-scan design for a sequential circuit and Figure 2.7 (b) shows a scan-flip-flop utilized by the full-scan design. In Figure 2.7 (a), every flip-flop (FF) is replaced with a scan FF (SFF) shown as Figure 2.7 (b). Each SFF can store an arbitrary value. For a sequential circuit designed by this technique, we can use a combinational test generation algorithm to generate a test pattern for the original circuit. Therefore, high fault efficiency can be achieved with short test generation time.

However, this method has the following disadvantages:

Area overhead: This approach adds extra DFT elements to FFs of the original sequential circuit. Therefore, compared with the area of the original sequential circuit, that of the sequential circuit designed by this method increases.

Delay overhead: This approach adds extra DFT elements to FFs of the original sequential circuit. Compared with the level of gates in the FF of the original sequential circuit, that in the SFF of the sequential circuit designed by this method increases. Therefore, the delay of the sequential circuit designed by this method increases.

Long test application time: This approach performs the scan-in and scan-out operation to control and observe the value in FFs of the circuit, respectively. Therefore, the scan-in and scan-out operations spend a lot of time.

No at-speed testing: This approach performs the scan-in and scan-out operations in the too slow operational speed compared with the normal operational speed of the original circuit. Therefore, test application at the rated speed of an original circuit can not be performed.

Althogh this method has some disadvantages, this method is usually utilized for easing LSI testing at many corporations. The reason is that this method can easily modify a original circuit and apply to any sequential circuit.

5. Summary

This chapter introduced the basic of test generation method for sequential circuits and processors, and design for testabiltiy method which is generally utilized for easing LSI testing. We discussed problems of the test for each method.

Chapter 3

Acceleration of Test Generation for Sequential Circuits Using Knowledge Obtained from Synthesis for Testability

1. Introduction

For general sequential circuits, it is difficult to achieve 100% fault efficiency in reasonable test generation time even for single stuck-at faults. The full-scan design is utilized to ease the test generation for sequential circuits[1]. However, we cannot perform *at-speed testing* for full-scan designed sequential circuits. To realize at-speed testing, an efficient test generation algorithm for sequential circuits, which generates tests for all the detectable faults and identifies all the untestable faults in reasonable test generation time, is necessary.

Most test generation algorithms for sequential circuits (e.g. HITEC[2], VERITAS[3], STALLION[5] and FASTEST[6]) employ a time frame expansion model of a sequential circuit. The time frame expansion model is a combinational circuit that simulates the exact behavior of the sequential circuit for a given number of time frames.

The HITEC is a well known test generator for sequential circuits. This method

consists of two phases. The first phase is the forward time processing phase in which a fault is excited and the resulting fault effect is propagated to a primary output. The second phase is the backward time processing phase which justifies the state required for activating the fault.

The VERITAS test generation method is an extension of the finite state machine (FSM) verification approach. This method constructs a product machine of a good FSM and its faulty version, and carries out reachability analysis by traversing the product machine. The information obtained by the reachability analysis is used to generate a test sequence. Although this simplifies generation of state justification sequences, it is not efficient to generate tests because it has to deal with huge product machines.

In this chapter, we propose a method of accelerating test generation for sequential circuits using the knowledge about a set of state justification sequences, the bound on the maximum length of state distinguishing sequences, the information about the valid states and the value of the reset state. We assume that circuits are given in FSM description. For circuits designed at register transfer level (RTL), controllers of the circuits are generally specified by FSM description. The proposed method is effective for such controllers. The sequential circuit is synthesized from a given FSM by a synthesis for testability (SFT) method proposed in this chapter which takes the features of our test generation method into consideration. The SFT method guarantees the existence of state distinguishing sequences of the specified length by making the given FSM reduced. Thus, the performance of the test generator is improved as it uses state justification sequences extracted from the completely specified state transition function of the FSM produced by the synthesizer. The proposed method can completely identify every fault in the circuit obtained by the proposed SFT method to be detectable or untestable. In our experiments, 100% fault efficiency is achieved for all the benchmark circuits in relatively short test generation time.

2. Preliminaries

In this chapter, we consider synchronous sequential circuits composed of combinational logic and D-type flip-flops (FFs). All the FFs are controlled by a single

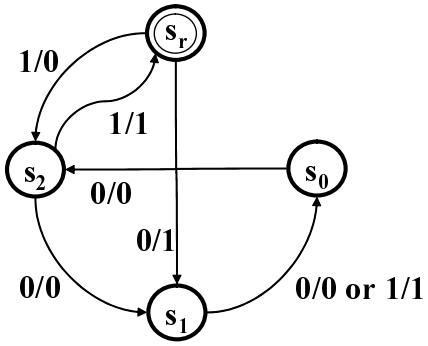


Figure 3.1. An incompletely specified finite state machine.

clock. We assume that a reset state is defined and a reset signal is available. We also assume that both the good and the faulty circuits can be put on the reset state by applying the reset signal. We consider the single stuck-at fault model but the faults on the clock lines, inside the FFs, and on the reset lines are not included in the fault set.

This paper deals with completely and incompletely specified Mealy-type FSMs. A Mealy-type FSM M is defined as a 6-tuple $\langle\ \Sigma,\ O,\ S,\ s_r,\ \delta,\ \lambda\ \rangle$. $\Sigma = \{x_0x_1 \ldots x_{n_i-1} \mid x_k \in \{0,1,X\},\ 0 \leq k < n_i\}$ is the set of *input vectors* and $O = \{z_0z_1 \ldots z_{n_o-1} \mid z_k \in \{0,1,X\},\ 0 \leq k < n_o\}$ is the set of *output vectors*, where X is the don't care, and n_i and n_o are the numbers of inputs and outputs, respectively. $S = \{s_r, s_0, s_1, \ldots, s_{n-2}\}$ is the set of states, where n is the number of states and s_r is the reset state. The functions δ and λ are the state transition function $S \times \Sigma \rightarrow S$ and the output function $S \times \Sigma \rightarrow O$, respectively. We assume that all the states defined in the FSM are reachable from the reset state s_r. For example, an incompletely specified Mealy-type FSM is shown in Figure 3.1.

A sequential circuit M_s composed of a combinational circuit part (CC) and FFs as shown in Figure 2 is synthesized from an FSM, where $x_0, x_1, x_2, \ldots, x_{n_i-1}$ are the primary inputs, $z_0, z_1, z_2, \ldots, z_{n_o-1}$ are the primary outputs and r is the reset input. We classify states represented by FFs of M_s into valid states and invalid states as defined below.

Definition 1 (Valid State and Invalid State) *A state s_i represented by the*

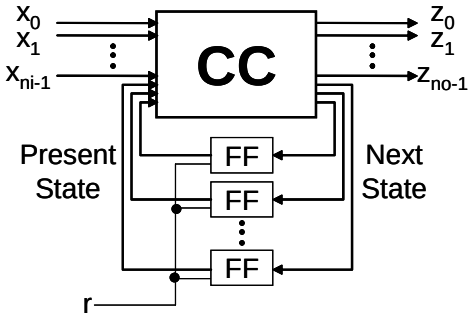


Figure 3.2. A sequential circuit M_s synthesized from an FSM.

FFs of a sequential circuit M_s is valid if s_i is reachable from the reset state of M_s. Otherwise, s_i is invalid. □

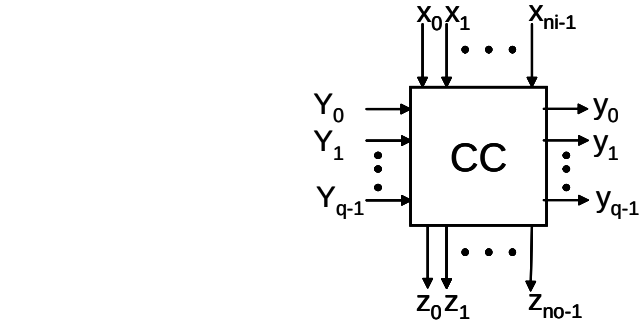
Definition 2 (State Distinguishing Sequence) *Let I be an input sequence of an FSM M. Let o_i and o_j be output sequences of I for M with initial states s_i and s_j, respectively. I is called a state distinguishing sequence with respect to the pair of states s_i and s_j if o_i and o_j are not identical.* □

Definition 3 (Reduced FSM) *An FSM is said to be reduced if every pair of states has at least one state distinguishing sequence.* □

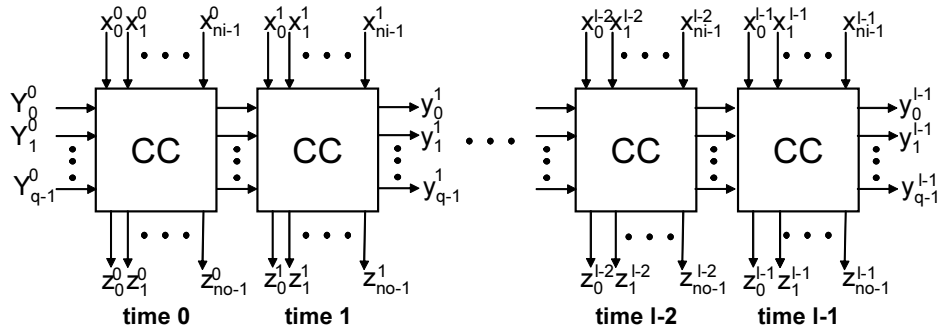
The proposed test generation method employs a time frame expansion model for the test generation.

Definition 4 (Time Frame) *A time frame is the combinational circuit extracted from a sequential circuit by treating its present state lines and next state lines as pseudo primary inputs and pseudo primary outputs, respectively.* □

Definition 5 (Time Frame Expansion Model) *A time frame expansion model of length l ($l \geq 2$) for a sequential circuit is a combinational circuit constructed by connecting time frames such that the pseudo primary outputs of a time frame i $(0 \leq i \leq l-2)$ is connected to the pseudo primary inputs of a time frame $i+1$.* □



(a)



(b)

Figure 3.3. A time frame of a sequential circuit M_s (a) and a time frame expansion model of M_s (b).

Examples of a time frame and a time frame expansion model are shown in Figure 3.3 (a) and (b), where $(Y_0^i, Y_1^i, \ldots, Y_{q-1}^i)$ and $(y_0^i, y_1^i, \ldots, y_{q-1}^i)$ are the pseudo primary inputs and the pseudo primary outputs of each time frame i, respectively.

3. Outline of the Proposed Method

The proposed method consists of an SFT method and a test generation method for sequential circuits synthesized by the SFT method. The SFT method synthesizes a sequential circuit to have the three specific characteristics from a given FSM. The proposed test generation method for the sequential circuit utilizes higher

level knowledge of its characteristics. By considering each characteristic, we can accelerate the fault excitation, the state justification and the error propagation, respectively. These three specific characteristics are the following.

Characteristic I: Any state in a sequential circuit synthesized from an FSM can be identified as either valid or invalid.

Characteristic II: There exists one to one correspondence between each state of the FSM and each valid state of the sequential circuit.

Characteristic III: For each pair of states in the sequential circuit, there exists a state distinguishing sequence. The maximum length of distinguishing sequences is k, which is a known constant.

The flow chart of the proposed method is shown in Figure 3.4. The area surrounded by the dotted line shows the SFT method and the outside area is our proposed automatic test pattern generation (ATPG) method. The italicized types in Figure 3.4 show knowledge extracted by the SFT. The knowledge is useful for the proposed test generation as follows.

***Information of valid states*:**
In a justification process of test generation, we don't need to justify a fault excitation state of a sequential circuit from the reset state if the fault excitation state is invalid. We can prune the search space of the justification process if we utilize the knowledge that helps to identify the fault excitation state as either valid or invalid. The knowledge "information of valid states" can be obtained since Characteristic I is satisfied. We can accelerate the whole fault excitation process during executing our proposed ATPG by reducing the number of calls of the fault excitation procedure by utilizing this information.

***A set of state justification sequences*:**
The state transition function of a given FSM is incompletely specified. The behavior of the FSM and the behavior of a sequential circuit synthesized from the FSM may be different, because the state transition function of the FSM is appropriately specified during the synthesis process and the state transition function of the sequential circuit becomes completely specified. We can justify the state of the sequential circuit easily if we can utilize the knowledge that helps

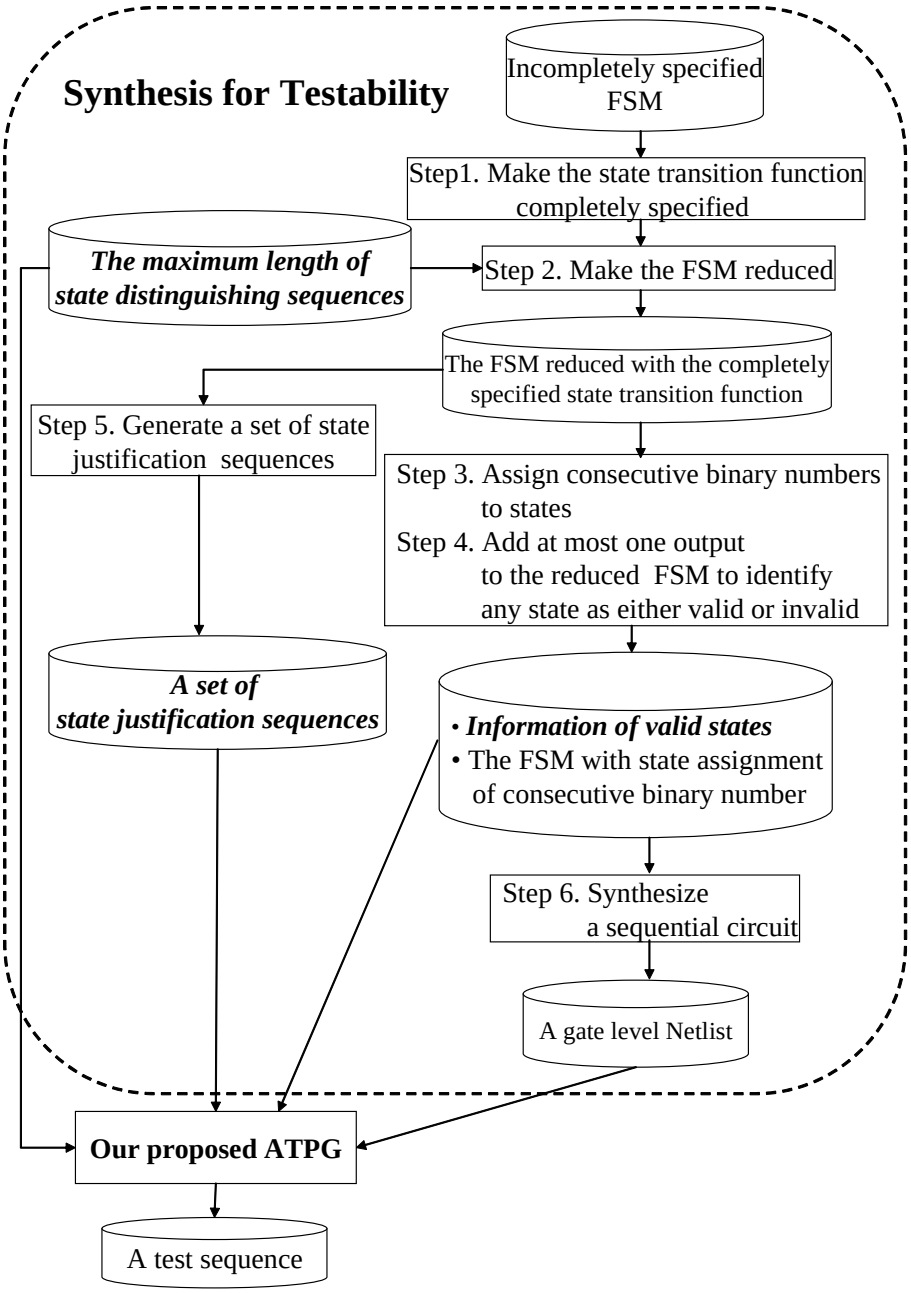


Figure 3.4. The flow chart of the proposed method.

to justify it by utilizing an input sequence, which is extracted from the FSM description, from the reset state to the excitation state. The knowledge "a set of state justification sequences" can be obtained since Characteristic II is satisfied. We can accelerate the state justification process using this information.

***The maximum length of state distinguishing sequences*:**
In general, we can't know the number of time frames which are required for propagating errors from the fault excitation frame to primary outputs of a sequential circuit in advance. However, we may limit the number of time frames expanded from the fault excitation frame if we have the knowledge of the number. The knowledge "the maximum length of state distinguishing sequences" is given by k since Characteristic III is satisfied. We can accelerate the error propagation process using this information.

4. Synthesis for Testability

In this section, we describe the proposed synthesis for testability (SFT) method for FSMs in detail. In the method, a sequential circuit which has three specific characteristics described in section 3 is synthesized from a given FSM. In order to synthesize a sequential circuit with such characteristics, a given FSM is modified as follows.

- Appropriate values are assigned to some of the coordinates which have don't care values in output vectors of the FSM.
- Extra outputs, if needed, are added to the FSM and appropriate values are assigned to them.

4.1 Formulation of SFT Problem

We formulate the SFT problem as an optimization problem as follows.

Input: An FSM with a reset state and the maximum length of state distinguishing sequences.

Output: A gate level netlist of a sequential circuit which has the three characteristics for the reset state, a set of state justification sequences and the number of valid states.

Objective: Minimization of the number of extra outputs.

4.2 Synthesis for Testability Algorithm

In this section, we propose a heuristic algorithm of the SFT since the minimization of the number of extra outputs is NP hard. The heuristic algorithm of the SFT consists of 6 steps as follows:

Step 1: Make the state transition function completely specified

Step 2: Make the FSM reduced

- **Step 2.1:** Try to generate state distinguishing sequences of length 1 for each pair of states of the FSM
- **Step 2.2:** Generate the k-partial state distinguishing tree in order to confirm that there exists a state distinguishing sequence of length less than or equal to k for each pair of states of the FSM and a state compatibility graph
- **Step 2.3:** Determine the number of extra outputs from the state compatibility graph

Step 3: Assign consecutive binary numbers to states in order to identify as either a valid or an invalid state

Step 4: Add an extra output to the FSM in order to guarantee existence of a state distinguishing sequence of length 1 for each pair of any valid state and any invalid state

Step 5: Generate a set of state justification sequences

Step 6: Synthesize a sequential circuit

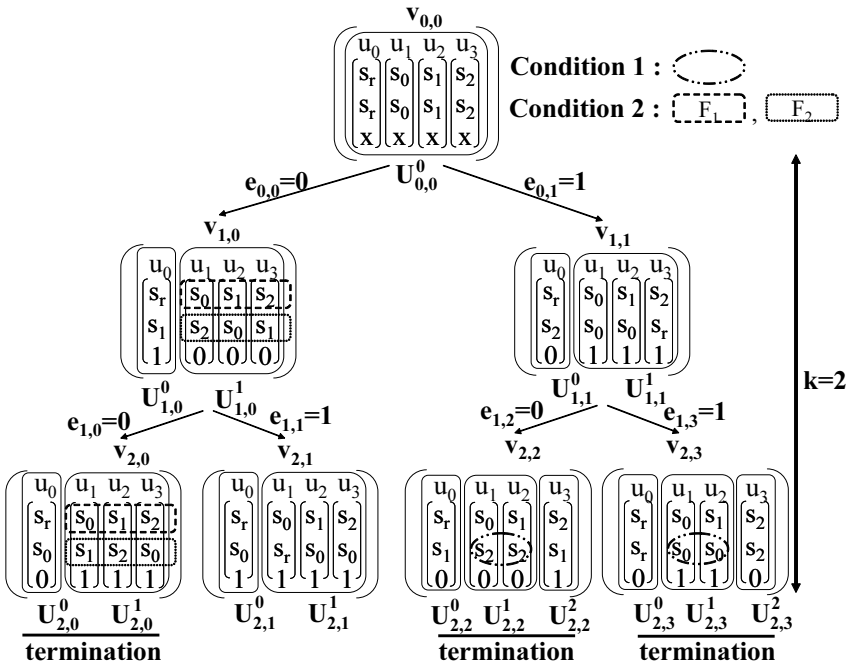


Figure 3.5. The 2-partial state distinguishing tree $T_2 = (V_{T_2}, E_{T_2})$.

In the heuristic algorithm, we use a k-partial state distinguishing tree and a state compatibility graph. We first define them as follows.

To clarify the discussion of state distinguishing sequences, we extend the definition of the successor tree defined in the literature [4] as follows.

Definition 6 (k-Partial State Distinguishing Tree) *Let M be an FSM. Let $T_k = (V_{T_k}, E_{T_k})$ be a tree of level k $(0 \leq k)$, where V_{T_k} is a set of nodes $\{v_{i,j_i} \mid 0 \leq i \leq k, 0 \leq j_i < |\Sigma|^i\}$ and E_{T_k} is a set of edges $\{(v_{i,j_i}, v_{i+1,j_i \cdot |\Sigma|+t}) \mid 0 \leq i < k, 0 \leq j_i < |\Sigma|^i, 0 \leq t < |\Sigma|\}$. An edge $(v_{i,j_i}, v_{i+1,j_i \cdot |\Sigma|+t})$ is also referred to as $e_{i,j_i \cdot |\Sigma|+t}$ and $\sigma_t \in \Sigma$ is associated with the edge. Let $\mathcal{U}$ be a set of states, which will be tried to be distinguished, of M and it is referred to as an initial uncertainty. Let U^p_{i,j_i} be a set of 3-tuples $\{u_n \mid 0 \leq n < |\mathcal{U}|\}$ and be associated with v_{i,j_i}, where p is the characteristic number and a 3-tuple $u_n \in U^p_{i,j_i}$ is composed of $s_n \in \mathcal{U}$, s_ℓ which is a state succeeded by applying the input sequence, which corresponds to a path from $v_{0,0}$ to v_{i,j_i}, and o_ℓ, which appears as the last output vector by*

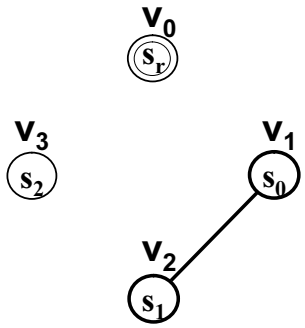


Figure 3.6. The state compatibility graph corresponding to Figure 3.5.

applying the input sequence to s_n, and is denoted in $< s_n, s_\ell, o_\ell >$. Here, u_n is called a distinguished state history (DSH). For each U^p_{i,j_i} of v_{i,j_i}, sets of DSHs of $v_{i+1,j_i\cdot|\Sigma|+t}$ are generated so that the DSHs are obtained by applying σ_t to M with the state of the second element of each $u_n \in U^p_{i,j_i}$ and these are classified into the sets where a set has the DSHs whose third elements are the same and they are different from the third elements of the DSHs in the other sets. The tree T_k is called a k-partial state distinguishing tree. □

Figure 3.5 shows the 2-partial state distinguishing tree $T_2 = (V_{T_2}, E_{T_2})$ for the FSM of Figure 3.1. Here, we suppose an initial uncertainty $\mathcal{U}$ of the FSM is a set of all the states of the FSM. Suppose a set of DSHs, $U^0_{0,0} = [u_0, u_1, u_2, u_3] = [< s_r, s_r, \mathcal{X} >, < s_0, s_0, \mathcal{X} >, < s_1, s_1, \mathcal{X} >, < s_2, s_2, \mathcal{X} >]$ is assigned to $v_{0,0} \in V_{T_2}$, where $\mathcal{X}$ is don't care vector such that all the bits of the output vector are don't care. By applying the vector $\sigma_1 = 1$ to each DSH of $v_{0,0}$, two sets $U^0_{1,1}$ and $U^1_{1,1}$, where $U^0_{1,1}$ is $[u_0] = [< s_r, s_2, 0 >]$ and $U^1_{1,1}$ is $[u_1, u_2, u_3] = [< s_0, s_0, 1 >, < s_1, s_0, 1 >, < s_2, s_r, 1 >]$, respectively, are associated with $v_{1,1}$. By applying the sequence $\sigma_1\sigma_0 = 10$ to each DSH of $v_{0,0}$, three sets $U^0_{2,2}$, $U^1_{2,2}$ and $U^2_{2,2}$, where $U^0_{2,2}$ is $[u_0] = [< s_r, s_1, 0 >]$, $U^1_{2,2}$ is $[u_1, u_2] = [< s_0, s_2, 0 >, < s_1, s_2, 0 >]$ and $U^2_{2,2}$ is $[u_3] = [< s_2, s_1, 1 >]$, respectively, are associated with $v_{2,2}$.

Definition 7 (State Compatibility Graph) *An undirected graph $G = (V_G, E_G)$, where $v \in V_G$ is a vertex corresponding to a state of an FSM and $e \in E_G$ is an*

edge corresponding to a pair of indistinguishable states of the FSM, is said to be a state compatibility graph. □

$\mathcal{U}$ is the initial uncertainty of an FSM. Let $D_s^{j_k}$ be a set of the distinguished states for $s \in \mathcal{U}$ of a leaf node $v_{k,j_k} \in V_{T_k}$ of a k-partial state distinguishing tree T_k obtained from the FSM where a distinguished state is a state in $\mathcal{U}$ except for s and is distinguishable from s. For all the leaf node of T_k, a set of states, which are distinguished from s, of $\mathcal{U}$ is obtained by the following formula: $\bigcup_{j_k=0}^{|\Sigma|^k} D_s^{j_k}$. The set of indistinguishable states of s is the complement of $\bigcup_{j_k=0}^{|\Sigma|^k} D_s^{j_k}$ for $\mathcal{U}$. We make the state compatibility graph based on pairs of indistinguishable states obtained from the above. Figure 3.6 shows the state compatibility graph corresponding to Figure 3.5. In this figure, indistinguishable states are s_0 and s_1.

Then, we describe the process for every step in detail.

Step 1: Let s_i be a state in M such that, there exist input vectors for which next states of the state are not specified in the state transition function. For each input vector $\sigma \in \Sigma$, which is not defined for a transition from the state s_i, of M, a state transition from s_i to s_i (i.e., a self-loop) in M for σ is added to the state transition function. An output vector o_i for the self-loop is added to the output function. All the bits of o_i are don't care. The FSM obtained in this step is referred to as M^α.

Step 2: To make every pair of states defined in M^α distinguishable, we perform the following three processes.

Step 2.1: For each input vector $\sigma \in \Sigma$ of M, we try to distinguish all the pairs of states s_i and s_j ($s_i \neq s_j$) of M. We perform the following two processes.

Step 2.1.1: Let o_i and o_j be output vectors of σ for M^α with s_i and s_j, respectively. We assign '0' or '1' to appropriate don't care bits of o_i in order to differentiate o_i and o_j if o_j is covered by o_i. Here, we define the relation between vectors a and b which have don't care values. We say that a covers b if $A \supset B$, where A and B are the sets of values represented by a and b, respectively.

Step 2.1.2: If o_i and o_j are the same and still have don't care bits, we assign '0' or '1' to some don't care bits of o_i and o_j to make o_i and o_j different. Let K be a set of such the same output vectors. Let $X(= x_0 x_1 \dots x_{n_X-1})$ be a vector composed of don't care bits in $\kappa \in K$, where n_X is the number of don't care bits in κ. The number of values represented by X is 2^{n_X}. If $|K| \leq 2^{n_X}$, the unique

value can be assigned to each κ. In this case, for each κ, we assign a unique value among 2^{n_X} to the don't care bits. If $|K| > 2^{n_X}$, we assign a value to each κ so that the number of the same output vectors is minimized. In this case, the consecutive binary number is cyclically assigned to the don't care bits in each κ. The FSM obtained in this step is referred to as M^β.

Step 2.2: We construct the k-partial state distinguishing tree to examine whether a pair of states s_i and s_j of M could be distinguishable by applying input sequences of length less than or equal to k to M^β with s_i and with s_j. We use the following two conditions of pruning for construction of the tree. Here, v and U^p are a current observed node of the k-partial state distinguishing tree and a set of DSHs of v whose third elements are the same and they are different from the third elements of DSHs in the other sets. Let V_q be the set of nodes on the path from the root to v. Let U_q^p be the set of DSHs of $v_q \in V_q$ whose the third elements are the same and they are different from the third elements of DSHs in the other sets. Let $F_1(U^p)$ and $F_2(U^p)$ be the set of first elements of all the DSHs in U^p and the set of the second elements of all the DSHs in U^p, respectively.

Condition 1: For each U^p of v such that $|U^p| \geq 2$, all the elements of $F_2(U^p)$ are the same.

Condition 2: There exists v_q such that for each U^p, whose number of DSHs is larger than 1, of v, there exists U_q^p, which satisfies $F_1(U_q^p) = F_1(U^p)$ and $F_2(U_q^p) = F_2(U^p)$, of v_q.

If v satisfies Condition 1 or Condition 2, v is a termination node. For example, in Figure 3.5, $v_{2,2}$ and $v_{2,3}$ satisfy Condition 1 and $v_{2,0}$ satisfies Condition 2. For $v_{2,2}$, all the elements of $F_2(U_{2,2}^1)$ are the same state s_2. For $v_{2,0}$, $F_1(U_{2,0}^1)$ and $F_2(U_{2,0}^1)$ of $v_{2,0}$ are equal to $F_1(U_{1,0}^1)$ and $F_2(U_{1,0}^1)$ of $v_{1,0}$ on the path from $v_{0,0}$ to $v_{2,0}$, respectively. In this case, the level of termination nodes is the same as the maximum level of the 2-partial state distinguishing tree.

Step 2.3: We construct the state compatibility graph obtained from the k-partial state distinguishing tree for representing all the indistinguishable state pairs of M^β.

For example, we obtain the state compatibility graph in Figure 3.6 from Figure 3.5. We can see that the indistinguishable states are s_0 and s_1 in the state compatibility graph.

We perform the following process in order to distinguish these indistinguishable states. Some outputs are added to M^β to distinguish all the indistinguishable state pairs. The problem to find the minimum number of additional outputs to distinguish all the indistinguishable state pairs is solved as a vertex coloring problem[7] of the state compatibility graph. The number of outputs to be added to M^β is obtained by the following formula:

$$n_a = \left\lceil \frac{\log_2 C}{|\Sigma|} \right\rceil,$$

where C is the number of colors obtained by solving the vertex coloring problem and n_a is the number of the additional outputs.

Let P be the set of values represented by the additional outputs. Let f_i be a mapping $\Sigma \stackrel{f_i}{\longmapsto} P$ such that $f_i \neq f_j, \quad {}^\forall i, j \mid 1 \leq i, j \leq C \wedge i \neq j$. For any $\sigma \in \Sigma$, the output function of M^β is changed so that the value of the additional outputs become $f_i(\sigma)$ for the state corresponding to each vertex, whose degree is more than or equal to 1, of the state compatibility graph. Thus, a state distinguishing sequence of length less than or equal to k is guaranteed for any state pair. The FSM obtained by this step is referred to as M^γ

Step 3: Let n_s be the number of states of the FSM M^γ. The number of FF, n_{ff}, in a sequential circuit synthesized from M^γ is equal to $\lceil \log_2 n_s \rceil$. The number of valid states of the circuit is equal to n_s and the number of invalid states, n_{iv}, is equal to $2^{n_{ff}} - n_s$. Binary numbers within the range of 0 to $n_s - 1$ are used for the state assignment of M^γ and binary numbers within the range of n_s to $2^{n_{ff}} - 1$ (if $n_{iv} \neq 0$) are used for values of the state variables of invalid states of the sequential circuit. The value assigned to the reset state s_r is referred to as n_r.

Step 4: To guarantee existence of a state distinguishing sequence of length 1 for each pair of any valid state and any invalid state of the sequential circuit synthesized by the SFT, one output is added to the FSM if n_{iv} is not equal to 0. This process means that a pair of any valid state and any invalid state is made distinguishable in order to realize Characteristic III. For a transition from a valid state to a valid state, '0' is assigned to the output. For a transition from an invalid state, '1' is assigned to the output. For a transition from a valid state to an invalid state, we have already considered in Step 1; all the transitions from

valid states are succeeded by valid states. The FSM obtained by this step is referred to as M^{ϵ}.

Step 5: For each valid state of M^{ϵ}, an input sequence to reach the state from the reset state is generated by a breadth-first search on the state transition graph of M^{ϵ}. By searching in a breadth-first fashion, the shortest input sequence is guaranteed for each state. The input sequence is called a state justification sequence.

A set of state justification sequences for all the valid states of M^{ϵ} is referred to as S_{si}.

Step 6: A gate level sequential circuit is synthesized from M^{ϵ} by a logic synthesis tool.

5. Test Generation Algorithm for Sequential Circuits

In this section, we describe the proposed test generation method that utilizes the knowledge of S_{si}: a set of state justification sequences, k: the maximum length of state distinguish sequences, n_s: the number of valid states, and n_r: the value of a reset state extracted by the SFT.

Our test generation method uses a time frame expansion model. A time frame expansion model has multiple faults because every time frame has the same single stuck-at fault. Therefore, our test generation method uses a 9 valued logic system [8][9] for the test generation to deal with multiple faults.

Figure 3.7 shows the flow chart of our test generation method for sequential circuits. The proposed test generation method consists of three processes: fault excitation, state justification and error propagation.

5.1 Fault Excitation

For a target fault, fault excitation finds an excitation vector which is assigned to primary inputs and pseudo primary inputs to produce errors and to propagate them to the primary outputs and/or the pseudo primary outputs of the fault excitation frame. The pseudo primary input part of an excitation vector is referred

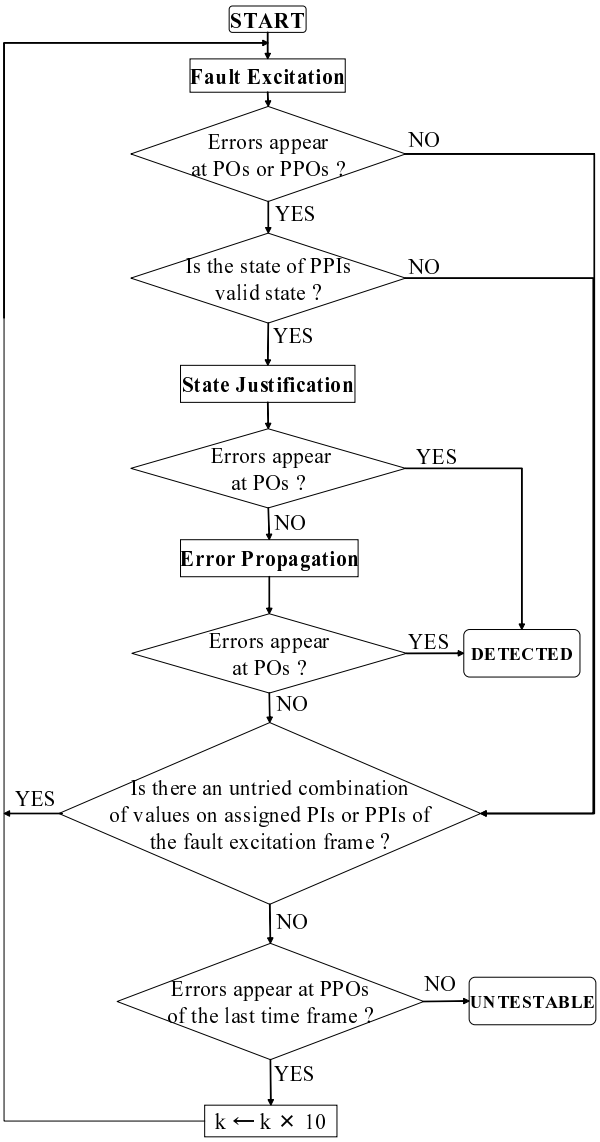


Figure 3.7. The flow chart of the proposed test generation method for sequential circuits.

to as an excitation state n_e. The number of valid states, n_s, helps generating a valid excitation vector which is an excitation vector whose excitation state is a valid state. If an excitation state is valid, the state may be justified from the reset state. However, if the excitation state is invalid, state justification is not required because the state cannot be justified from the reset state. Hence, the proposed method can prune a part of search space of a test generation. This search space pruning is realized by comparing n_s with n_e. If n_e is less than n_s, the excitation state is valid. Otherwise, the state is invalid. This feature saves a large amount of time for trying to generate invalid excitation vector and trying to justify the invalid excitation state. If there exists no valid state to excite the fault, the fault is proved untestable.

5.2 State Justification

Once an excitation vector is found, state justification is performed. The excitation state must be justified for both the fault-free circuit and the faulty circuit. We have a set of state justification sequences, S_{si}, for the fault-free circuit. The fault-free state justification can be easily done by choosing the state justification sequence for the excitation state from S_{si}.

No backtracking is required and no failure can occur in this step. The next step is to confirm if the fault-free state justification sequence is also valid for the faulty circuit. This is confirmed by fault simulation using the fault-free state justification sequence and observing if any invalidation occurs. Figure 3.8 shows an example of an invalidation. An invalidation means that a state transition of the faulty circuit is different from the fault-free circuit. If an invalidation occurs, the state justification sequence cannot justify the given excitation state because the state justification sequence is not guaranteed to work under the faulty circuit. However, if an invalidation occurs, some error must appear on the pseudo primary outputs of some frame (we call this an actual excitation state) between the reset frame and the fault excitation frame. We try to propagate errors from the actual excitation state. If some error appears on the primary outputs between the reset frame and the fault excitation frame, the fault is detected.

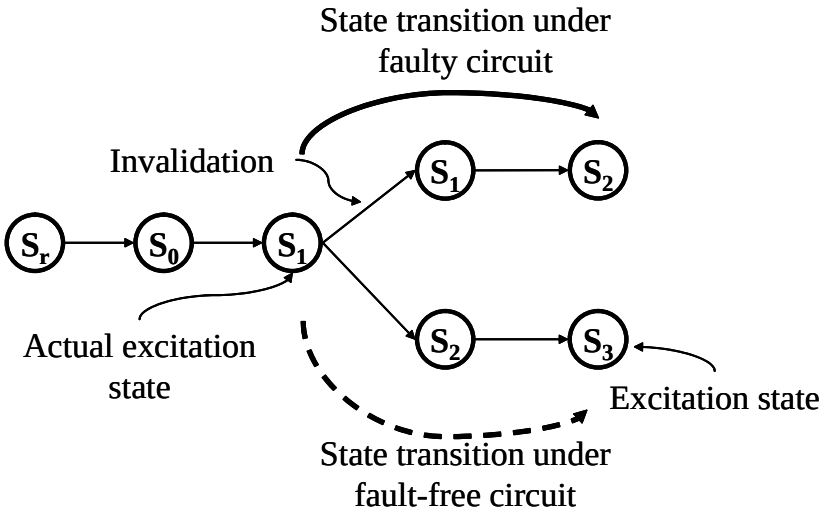


Figure 3.8. An example of an invalidation.

5.3 Error Propagation

If a fault is not identified as detected or untestable by the first two processes, error propagation is performed. Time frames of length k are added to the (actual) fault excitation frame. The error propagation process determines primary input values of the expanded time frames to propagate an error to a primary output. This process may not propagate any error to any primary output and any pseudo primary output because errors may be masked by the multiple faults within the added k time frames. In this case, we try to search a different excitation state by returning to the fault excitation process. On the other hand, any error is not propagated to any primary output but any error is propagated to some pseudo primary output of the last time frame. This is because k-state distinguishing sequence is not guaranteed for faulty circuit. Therefore, in order to make error propagation complete, the number of time frames expanded from the fault excitation frame has to be increased (e.g., $k = k \times 10$, where this number 10 might be changed empirically). We perform fault excitation again.

6. Experimental Results

Table 3.1 and Table 3.2 show characteristics of the MCNC FSM benchmarks[10] and the results of SFT. All the experiments except for the proposed SFT were performed on a SUN Blade 2000 (CPU 1GHz $\times$ 2) with 8GB memory. The experiments for the proposed SFT were performed on a PC/AT machine (CPU Athlon 3000+) with 1GB memory. Design Compiler (Synopsys) is used as a logic synthesis tool for the Step 6 of the proposed SFT method. The number of benchmarks is 53. For all the benchmarks, the proposed method could perform until Step 5. However, Design Compiler was unable to perform Step 6 for 14 benchmarks because of restrictions on Design Compiler. One of the restrictions is that the size of FSM descriptions which Design Compiler can read is limited.

All the benchmarks shown in Table 3.1 and Table 3.2 were synthesized by using some optimization options which optimize the area and the delay of a circuit. The first four columns give the benchmark name and the numbers of primary inputs, primary outputs and states, respectively. The column "k" gives the maximum length of the state distinguishing sequences. The columns "#EO," "HOH" and "#MLG" give the results of the proposed SFT. The column "#EO" denotes the number of extra outputs added to each benchmark. The column "HOH" denotes the hardware overhead which is the ratio of the area of the sequential circuit synthesized by the proposed SFT to that of the original sequential circuit synthesized by the ordinary synthesizer. The subdivided columns "Orig." and "Prop." in "#MLG" are the maximum level of gates in the original sequential circuit and that of gates in the sequential circuit obtained by the proposed SFT, respectively.

In this experiment, the maximum number of outputs added to the FSMs is two: one for distinguishing between valid and invalid states and the other for making the given FSM reduced. The number of extra outputs decreases when the FSM is reduced by only assigning values to the don't cares in the output vectors of the FSM.

The average hardware overhead is 30.18%. However, the hardware overhead of the circuit 'pma' is more than 300%. It is considerable that the logic synthesis tool may not be able to simplify logic because we assign logic values to coordinates with don't cares in input vectors and output vectors to make a given FSM reduced.

However, a hardware overhead may be able to be reduced by carrying out recoding of input vectors and output vectors. Hence, still there is room for research on how to assign coordinates with don't cares in input vectors and output vectors during SFT.

The maximum time spent for the proposed SFT method is about twenty minutes. For small circuits (ex. lion, bbara, bbsse, bbtas and so on), the time spent for the SFT is 0.1 second or less. Notice that since we have no way to obtain the optimal length of state distinguishing sequences, in the experiments, we determined a practical length k by running trials as follows. For an FSM, we set a time limit to determine k for the trials. For k, we tried recursively to create k-partial state distinguishing tree and to confirm that there exists a state distinguishing sequence of length less than or equal to k for each pair of states of an FSM by running the procedure of Step 2.2 in the proposed SFT method. If the whole run time of the trials exceeds the time limit or the number of distinguishable states does not increase compared with that of the previous trial, we take k of the previous trial. Otherwise, we increment k by one and perform the next trial. In the experiments, we incremented k from one and set the time limit one hour for each benchmark circuit. To find a way to do that is included in our future work.

For almost all the circuits, the maximum level of gates in the sequential circuits obtained by the SFT is less than that in the sequential circuit synthesized from the FSM without using the SFT. However, the maximum level of gates in some circuits synthesized by the proposed SFT becomes about twice compared with that synthesized by the ordinary synthesizer. In this case, the performance of these circuits degrades from the original sequential circuits. Typically, the maximum level of gates in these circuits is much less than that of the data path in a VLSI. We believe that the performance of these circuits depends on the performance of a data path and the delay of a circuit is able to be absorbed on the data path side.

Table 3.3 and Table 3.4 show the test generation results for three different methods. Method 1 applies TestGen (Synopsys) to the original sequential circuit synthesized from the FSM without using the proposed SFT. Method 2 applies TestGen to the sequential circuit obtained by the proposed SFT. Method 3 applies

Table 3.1. Characteristics of FSM benchmarks and results of SFT.

Circuit	#Input	#Output	#State	k	#EO	HOH (%)	#MLG	
							Prop.	Orig.
bbara	6	2	10	3	2	3.36	14	17
bbsse	9	7	13	2	1	43.03	28	34
bbtas	4	2	6	5	1	3.45	11	8
beecount	5	4	7	2	1	12.82	27	11
cse	9	7	16	1	0	27.3	22	30
dk14	5	5	7	1	1	-0.28	22	26
dk15	5	5	4	1	0	0	18	18
dk16	4	3	27	2	1	9.76	48	39
dk17	4	3	8	1	0	0	15	15
dk27	3	2	7	2	1	-1.28	12	8
ex1	11	19	20	1	1	69.61	34	35
ex3	4	2	10	2	1	35.15	18	23
ex4	8	9	14	1	1	16.36	16	25
ex5	4	2	9	2	1	39.56	17	18
ex6	7	8	8	1	0	-1.69	17	26
keyb	9	2	19	1	1	44.6	36	29
kirkman	14	6	16	1	0	104.34	45	44
lion	4	1	4	2	0	0	9	12
lion9	4	1	9	6	1	14.29	17	33

Table 3.2. Characteristics of FSM benchmarks and results of SFT. (cont.)

Circuit	#Input	#Output	#State	k	#EO	HOH (%)	#MLG	
							Prop.	Orig.
mc	5	5	4	1	0	0	6	8
opus	7	6	10	1	1	28.52	21	28
planet	9	19	48	2	1	24.76	41	33
planet1	9	19	48	2	1	24.76	41	33
pma	10	8	24	1	1	303.04	100	59
s1	10	6	20	1	1	-5.83	47	67
s1488	10	19	48	2	1	-0.71	48	88
s1494	10	19	48	2	1	-20.73	42	73
s208	13	2	18	3	2	2.41	23	24
s27	6	1	6	2	2	4.4	14	15
s298	5	6	218	6	2	11.68	107	146
s386	9	7	13	2	1	-1.47	29	41
s420	21	2	18	1	2	56.02	14	24
styr	11	10	30	1	2	62.42	41	30
sse	9	7	16	1	0	77.27	76	48
tma	9	6	20	1	1	120.75	5	8
tbk	8	3	32	1	1	36.39	46	67
tav	6	4	4	3	0	0	48	50
train11	4	1	11	2	2	28.83	26	19
train4	4	1	4	2	0	4.08	8	17

the proposed test generation method to the sequential circuit obtained by the proposed SFT. The column "TGT [s]" denotes the time, in seconds, which was spent on the test generation excluding the fault simulation time. The subdivided columns "m1," "m2" and "m3" denote the method 1, the method 2 and the method 3, respectively. The column "TTGT [s]" denotes the time, in seconds, which is the sum of "TGT" and the fault simulation time. The fault simulation time is calculated by "TTGT" - "TGT." The time '> 10h' in the columns "TGT [s]" and "TTGT [s]" means that the test generation did not finish within 10 hours.

All the methods performed the equivalent fault analysis and the fault simulation which is implemented in TestGen. Since our test generator does not have the desired fault simulation capability, the method 3 requires a call to the external fault simulator. Therefore, in order to compare the test generation time of the proposed method with that of TestGen on equal terms, we performed the fault simulation for the test sequence obtained by the test generation.

The total test generation time for each benchmark of the method 3 is shorter than that of the other methods. For some benchmarks, the total test generation time of the method 2 is longer than that of the method 1. For all the experiments of the method 3, k was not increased. In other words, the error propagation process of the method 3 performed completely within k frames expanded for the process where k was given as the input of our proposed ATPG. We believe that the method 3 can effectively use the knowledge obtained by the SFT but the method 2 cannot effectively use it. The average total test generation time of the method 1, that of the method 2 and that of the method 3 are 9694.06 (s), 3371.99 (s) and 2.97 (s), respectively. The actual average total test generation time of the method 1 and the method 2 will be longer because these methods did not achieve 100% fault efficiency for some circuits. The method 3 identified all the untestable faults within reasonable time.

The columns "FC [%]," "FE [%]," and "TSL" are the fault coverage, the fault efficiency and the length of the test sequence, respectively. The method 3 can achieve 100% fault efficiency for all the benchmarks within reasonable time and the time is shorter than both the test generation time of the method 1 and that of the method 2. However, the method 1 and the method 2 did not achieve 100%

fault efficiency for several these benchmarks. Particularly, both the method 1 and the method 2 for 's298' did not achieve 100% fault efficiency within 10 hours. The proposed method can perform faster test generation than the conventional method for benchmarks.

The test sequence length of the method 3 is shorter than that of the other methods for about a half of benchmarks. There are some cases where the test sequence length of the method 3 is longer than that of the other methods, this is because TestGen uses techniques of test sequence compression.

7. Summary

In this chapter, we proposed a method for high speed test generation for sequential circuits with some specific characteristics. Such a sequential circuit can be synthesized from a given FSM by the synthesis for testability (SFT) method which takes the features of our test generation method into consideration. We accelerated test generation for sequential circuits by utilizing the knowledge that consisted of a set of state justification sequences, the maximum length of state distinguishing sequences, the number of valid states, and the value of the reset state extracted by the SFT. The experimental results show that the proposed method can achieve 100% fault efficiency in shorter test generation time compared to a state of the art test generator.

Table 3.3. Test generation results for each method.

Circuit	TGT [s]			TTGT [s]			FC [%]			FE [%]			TSL		
	m1	m2	m3	m1	m2	m3	m1	m2	m3	m1	m2	m3	m1	m2	m3
bbara	> 10h	2.12	0.09	> 10h	2.26	0.24	94.95	95.63	95.63	96.46	100.00	100.00	891	690	486
bbsse	1.70	2.03	0.28	1.83	2.23	0.44	98.27	97.46	97.46	100.00	100.00	100.00	486	762	801
bbtas	1.60	2.89	0.03	1.68	3.03	0.11	98.61	95.24	95.24	100.00	100.00	100.00	276	318	216
beecount	0.26	0.28	0.02	0.38	0.38	0.14	97.53	97.80	97.80	100.00	100.00	100.00	342	285	270
cse	2.28	1.69	0.21	2.49	1.87	0.40	100.00	100.00	100.00	100.00	100.00	100.00	1119	915	924
dk14	0.32	0.29	0.02	0.45	0.41	0.13	98.61	98.21	98.21	100.00	100.00	100.00	300	288	270
dk15	0.09	0.15	0.00	0.21	0.28	0.11	100.00	100.00	100.00	100.00	100.00	100.00	108	108	117
dk16	> 10h	92.75	0.63	> 10h	93.06	0.87	98.29	98.17	98.17	99.74	100.00	100.00	1323	1368	951
dk17	0.13	0.12	0.02	0.22	0.24	0.16	100.00	100.00	100.00	100.00	100.00	100.00	171	180	171
dk27	0.10	0.12	0.00	0.21	0.21	0.11	95.59	94.44	94.44	100.00	100.00	100.00	81	75	54
ex1	5538.05	> 10h	1.02	5538.23	> 10h	1.32	97.57	98.52	98.52	100.00	99.89	100.00	906	1026	990
ex3	10.53	0.53	0.06	10.65	0.65	0.19	96.46	97.27	97.27	100.00	100.00	100.00	372	399	330
ex4	0.82	0.31	0.03	0.97	0.39	0.15	97.08	97.08	97.08	100.00	100.00	100.00	660	306	372
ex5	5639.33	3.86	0.04	5639.44	3.96	0.16	95.65	95.20	95.20	100.00	100.00	100.00	417	390	294
ex6	0.19	0.16	0.01	0.30	0.27	0.14	100.00	100.00	100.00	100.00	100.00	100.00	240	183	201
keyb	10.39	30.20	1.12	10.59	30.42	1.35	97.81	97.38	97.38	100.00	100.00	100.00	1050	1104	1188
kirkman	1.13	1.51	0.61	1.34	1.80	0.96	100.00	100.00	100.00	100.00	100.00	100.00	1437	1698	1839
lion	0.08	0.11	0.00	0.16	0.22	0.08	100.00	100.00	100.00	100.00	100.00	100.00	120	120	81
lion9	> 10h	36.69	0.05	> 10h	36.81	0.16	95.71	95.62	95.62	98.57	100.00	100.00	408	456	432

Table 3.4. Test generation results for each method. (cont.)

Circuit	TGT [s]			TTGT [s]			FC [%]			FE [%]			TSL		
	m1	m2	m3	m1	m2	m3	m1	m2	m3	m1	m2	m3	m1	m2	m3
mc	0.09	0.09	0.00	0.19	0.20	0.07	100.00	100.00	100.00	100.00	100.00	100.00	87	87	51
opus	7.72	7.19	0.10	7.85	7.30	0.22	98.76	96.50	96.50	100.00	100.00	100.00	408	456	498
planet	1562.30	1160.20	2.26	1562.83	1160.95	3.22	98.96	98.97	98.97	100.00	100.00	100.00	2721	3426	3963
planet1	1562.30	1160.20	2.26	1562.83	1160.95	3.22	98.96	98.97	98.97	100.00	100.00	100.00	2721	3426	3963
pma	> 10h	10185.70	25.71	> 10h	10187.76	27.88	98.83	99.44	99.44	99.61	100.00	100.00	1248	4404	4527
s1488	1282.41	4544.74	1.18	1283.00	4545.43	1.79	98.74	99.01	99.01	100.00	100.00	100.00	2787	3693	3129
s1494	1965.04	6027.33	1.37	1965.60	6027.95	2.07	98.71	98.87	98.87	100.00	100.00	100.00	2676	2856	3375
s1	> 10h	170.99	1.01	> 10h	171.23	1.22	97.02	96.84	96.84	99.46	100.00	100.00	1182	1542	972
s208	> 10h	2.45	0.04	> 10h	2.55	0.18	95.00	94.44	94.44	98.57	100.00	100.00	603	612	510
s27	0.27	0.19	0.12	0.36	0.28	0.21	91.03	90.12	90.12	100.00	100.00	100.00	102	102	81
s298	> 10h	> 10h	48.01	> 10h	> 10h	57.09	95.75	95.75	92.12	96.12	92.56	100.00	29325	22422	14052
s386	4.60	2.44	0.16	4.78	2.60	0.30	97.22	96.27	96.27	100.00	100.00	100.00	663	795	567
s420	> 10h	3.66	0.08	> 10h	3.78	0.22	95.00	96.12	96.12	98.57	100.00	100.00	651	858	1020
sse	377.75	1.61	0.44	377.91	1.79	0.63	97.17	97.26	97.26	100.00	100.00	100.00	543	645	747
styr	18.30	30.22	4.25	18.79	30.92	4.95	99.54	99.72	99.72	100.00	100.00	100.00	2367	2460	1783
tav	0.09	0.11	0.01	0.18	0.19	0.11	100.00	100.00	100.00	100.00	100.00	100.00	81	81	120
tbk	> 10h	7.47	2.99	> 10h	8.12	3.08	98.85	100.00	100.00	98.85	100.00	100.00	1200	2586	2025
tma	> 10h	> 10h	1.17	> 10h	> 10h	1.69	99.02	98.92	98.92	99.41	99.91	100.00	789	1287	1233
train11	55.01	0.71	0.05	55.13	0.86	0.18	97.31	97.06	97.06	100.00	100.00	100.00	342	300	321
train4	0.13	0.13	0.00	0.23	0.22	0.09	100.00	100.00	100.00	100.00	100.00	100.00	120	138	99

Chapter 4

Design for Testability of Software-Based Self-Test for Processors

1. Introduction

In recent years, it has been essential that processors with high performance and rich functionality have accurate and at-speed testing. Though the full-scan approach is commonly used due to its simplicity, it induces performance penalty, area overhead and excessive power consumption. The hardware built-in self-test (BIST), which is one of the other widely used techniques, applies pseudo-random test patterns to modules on the circuit at the normal operational speed. However, design modifications are required to make a circuit to be BIST-ready, and involve large amount of manual effort. The BIST also induces area overhead. Furthermore, an application of random patterns results in excessive power consumption.

A number of approaches [11-17] have been proposed for software-based self-test (SBST) as a promising approach to resolve the above problems. In SBST, we test a processor by executing a sequence of instructions called a test program. A processor can be tested by communicating with the memory, and thus it enables at-speed testing. We use communication between the memory and the outside ATE as pre- and post-processes of the execution of the test program.

Some methods among the SBST methods generate a test program based on

test program templates targeting structural faults to achieve the high fault coverage [13-17]. In this approach, gate-level test generation is applied to generate test patterns for each module under test (MUT) of a processor (MUT test generation), and a test program is synthesized from test patterns (test program synthesis), where a test program justifies the test pattern from the memory to the MUT and propagates the test response from the MUT to the memory. To guarantee the test program synthesis, test program templates are used. A test program template is an instruction sequence with unspecified operands that delivers a test pattern to an MUT and observes the test response. The approach extracts constraints on the input and output of the MUT from each template, and applies test generation for the MUT under the constrains. In this approach, we can easily synthesize a test program from a test pattern for the MUT. However, the justification and observation parts consider only behavior of a fault-free processor and do not consider behavior of a faulty processor, and such parts might not work as expected. In this case, some faults detected by a test pattern for an MUT may not be detected by the synthesized test program. We call such a phenomenon "error masking."

In this chapter, we propose a design for testability (DFT) method that completely resolves the problem of error masking for any test program generated by the template-based SBST approach for the stuck-at fault. We show a sufficient condition for avoiding error masking and propose the DFT method which satisfies the sufficient condition. In the experimental results, we show the hardware overhead caused by the proposed DFT method and results of the execution of a test program for processors before/after applying the proposed DFT method. The proposed method enables at-speed testing.

This chapter is organized as follows. In sections 2 and 3, we show a processor model and test program generation using templates, respectively. In section 4, we analyze error masking and define template level fault efficiency. In section 5, we propose a sufficient condition for avoiding error masking. In section 6, we propose a DFT method of SBST for processors and the experimental results are shown in section 7. Finally, the paper is concluded in section 8.

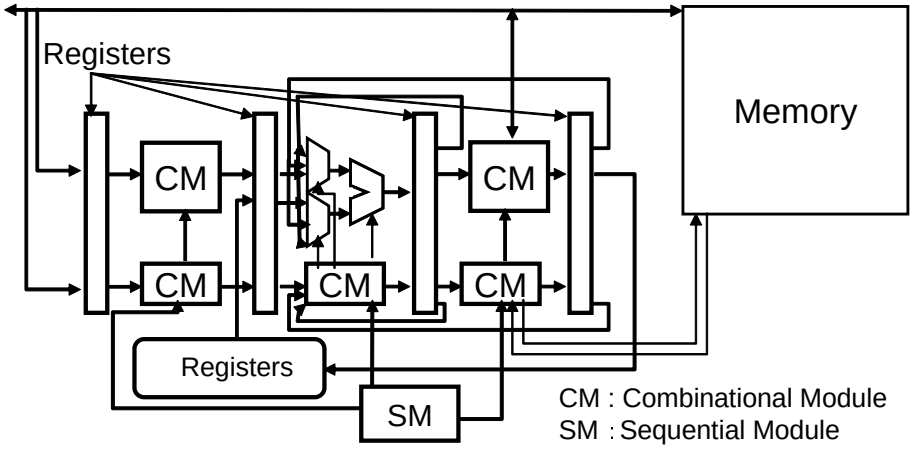


Figure 4.1. An example of a processor.

2. Processor Model

A processor is specified by register transfer level (RTL) description. Figure 4.1 illustrates an example of a processor. A processor consists of combinational modules such as arithmetic logic unit (ALU) or multiplexer (MUX), sequential modules such as a controller specified as a finite state machine, signals where a signal is referred to as an RTL signal that connects the modules, and buses. A bus considered to be a tri-state bus[21]. For a fault-free processor, we also assume the following about tri-state buses.

(1) Two or more inputs of a tri-state bus are not activated simultaneously.

(2) Each output of the tri-state bus has a masking circuit that generates a logic value ('0' or '1'), and an output of a tri-state bus is masked into some specific logic value if any input of the tri-state bus is not activated.

A processor is assumed to be synthesized while preserving the hierarchy of the modules, and therefore each module can be identified in a gate-level description.

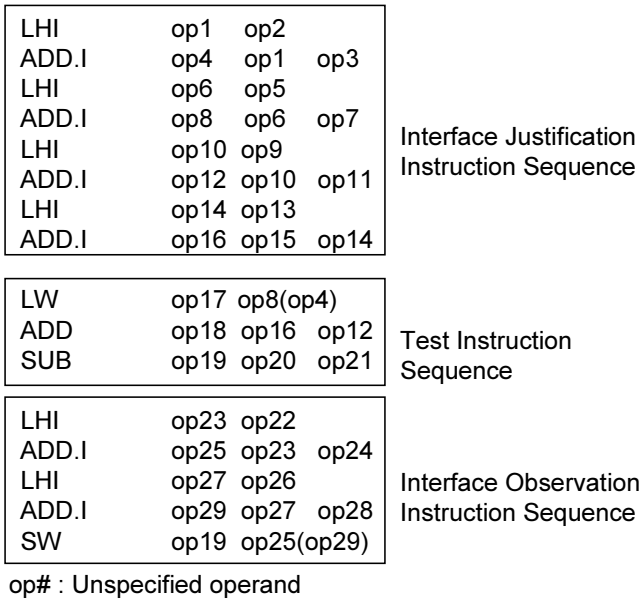


Figure 4.2. An example of a template.

3. Test Program Generation based on Templates

We first explain test program generation using test program templates. In the rest of this chapter, we call a test program template a template. Figure 4.2 illustrates an example of a template, which consists of three sequences: a justification instruction sequence, a test instruction sequence and an observation instruction sequence. A justification instruction sequence is utilized for justifying test patterns to registers which are adjacent to inputs of an MUT. A test instruction sequence applies test patterns to the MUT and propagates the test response to registers that are adjacent to outputs of the MUT or the memory. An observation instruction sequence propagates the test response stored in registers to the memory. For each template, we extract constraints about the input space and the output space of the MUT and perform the MUT test generation under constraints.

Figure 4.3 illustrates a model of an MUT test generation under constraints extracted from a template. The input and output constraint are extracted from

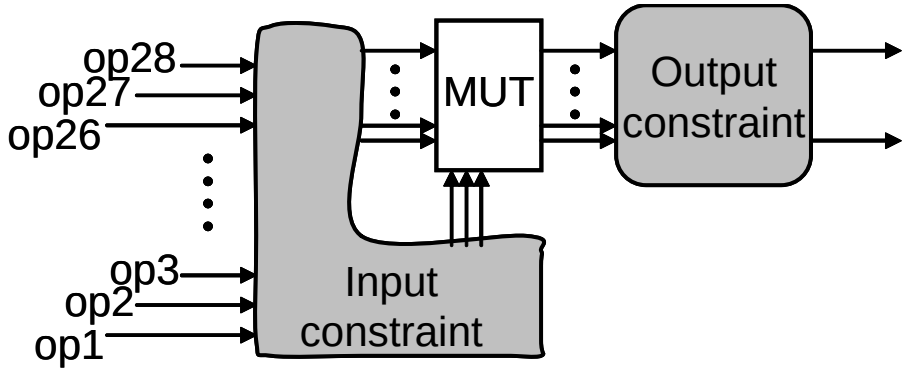


Figure 4.3. A model of an MUT test generation.

a justification instruction sequence and the test instruction sequence, and the test instruction sequence and an observation sequence, respectively. The both constraints represent the behavior which includes the operation of the MUT. If the processor is fault-free, a test program synthesized from test patterns generated by using the model in Fig. 4.3 can justify test patterns to the inputs of the MUT and observe the test responses. However, in the case of the faulty processor, the both constraints show the incorrect behavior because the MUT appears and is utilized during the executing of the test program. Therefore, the test program synthesized from the test generation model might not justify test patterns to the inputs of the MUT and observe the test responses. Throughout the paper, we call a test program obtained by the test program generation *template-based test program* and restrict fault model to a stuck-at fault.

4. Error Masking

4.1 Template Level Fault Efficiency

In the test program generation method using templates, justification instruction sequences, and observation instruction sequences are generated only in consideration of the behavior of the fault-free processor. When applying a test program to the faulty processor, errors may appear during the execution of these sequences.

Therefore, we cannot guarantee that the test program justifies test patterns of the MUT, or observe the test response. This means that some faults detected in the MUT test generation may not be detected by the test program synthesized from the test. We call this phenomenon "error masking." In this chapter, we define template level fault efficiency (FE_T) as measure to evaluate error masking as follows:

$$FE_T = \frac{F_{TP}}{F_{MT}},$$

where F_{MT} is the number of faults detected in the MUT test generation and F_{TP} is the number of faults detected by the test program among the F_{MT} faults. A template level fault efficiency of 100% means that there is no error masking.

4.2 Analyzing Error Masking

Figure 4.4 illustrates examples of error masking using a time frame expansion model of the execution of a test program, where each time frame corresponds to one clock. In the time frame expansion model, time frames that apply test patterns to the MUT are called "test frame", while time frames before the test frame are called "justification frame" and time frames after the test frame are called "observation frame". Modules M_J, M_T and M_O denote the same MUT which appears repeatedly in each time frame. The following phenomena can be considered.

Figure 4.4 (a) illustrates an example of error masking induced by an unknown value. In this figure, we describe the value of an RTL signal as "the value in fault-free circuit/value in faulty circuit." Values in gray field denotes the propagated values which include unknown values. In Fig. 4.4 (a), unknown values reach an input of M_J and the specified values reach another input. If M_J operates correctly, unknown values do not appear the output of its module. However, if M_J operates incorrectly under the effect of the fault, values '11/1X' which include an unknown value is propagated through its module. Therefore, values '01/0X' which include an unknown value are propagated to an input of M_T at the test frame and fail to activate the fault at the test frame.

Figure 4.4 (b) illustrates an example of error masking induced by a cycle. In Fig. 4.4 (b), the fault is excited at M_J of the justification frame and errors appear

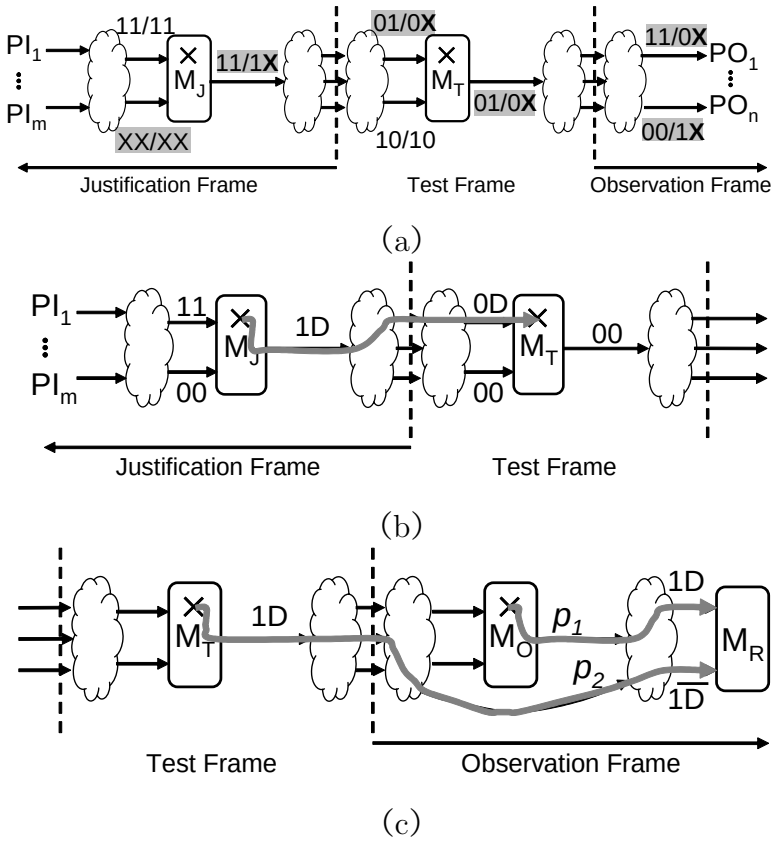


Figure 4.4. Examples of error masking : (a) unknown values are propagated to RTL signals; (b) errors reach the MUT; (c) errors are propagated to two RTL signals and meet at some module in some frame.

the output of its module. The errors reach inputs of M_T in the test frame. A fault of M_T is not excited because incorrect test patterns reach to inputs of its module. In this case, there is a cycle that includes the MUT.

Figure 4.4 (c) illustrates an example of error masking induced by the convergence of errors. In Fig. 4.4 (c), the same fault in M_T and M_O is excited, and errors are propagated to two paths p_1 and p_2 meet at some module M_R. The multiple errors mask each other in M_R in the observation frame and no error is propagated to the output of M_R. The phenomenon in Figure 4.4 (c) might occur if there are reconvergent paths between the MUT and some module.

5. Sufficient Condition for Avoiding Error Masking

In this section, we show a sufficient condition for avoiding error masking. In the sufficient condition, we utilize a reconvergent path.

Definition 8 (Reconvergent path) *Let M and M' be modules. Paths p_i and p_j are reconvergent paths from M to M' if both p_i and p_j are paths from M to M', and p_i and p_j share no module except M and M'.*

We give the following two assumptions for a test program.

- The value stored in the memory cell referred to during the execution of the test program is known. That is, in the fault-free processor, unknown values are not propagated from the memory to the processor.
- The memory cell referred to during the execution of the test program is initialized to a different value from the expected value, which will be stored in the memory cell during the execution of the test program.

In this chapter, we consider a fault is detected in the following cases.

(A1) No value or an incorrect value is stored in some memory cell where the test program should write some expected value.

(A2) The test program fails to read a value from a memory cell designated to be read in the test program.

The second condition (A2) holds if the memory address lines are observable. However, even if they are not observable, we consider such incorrect behavior would cause some observable errors.

To guarantee that the proposed method can achieve 100% template level fault efficiency, we show a sufficient condition for a processor such that error masking does not occur during the execution of the template-based test program. In the proof of theorem 1, we consider four values '0', '1', 'X' (uninitialized value) and 'Z' (high-impedance) for each bit of an RTL signal line. We consider 'X' and 'Z' to be unknown value, and an error of an RTL signal line means that at least one bit has different known values.

Theorem 1 *For any template-based test program, error masking does not occur during the execution of the test program if a processor satisfies the following four conditions.*

(1) *Each register is initialized at the beginning of the execution of the test program.*

(2) *All the control signals of each tri-state bus and its masking circuits are observable.*

(3) *For each cycle, at least one RTL signal on the cycle is observable.*

(4) *For each pair of reconvergent paths, at least one RTL signal on the two paths is observable.*

Proof:

Let f be a stuck-at fault detected by an MUT test generation in template-based test program generation. We consider the execution of a test program for f. Let M be a module with f.

First we show that f is detected or the value of any RTL signal of the processor is known.

Since all the registers are initialized to known values at the beginning from condition (1), if some signal has an unknown value, it comes from the outside or is generated at the inside of the processor. If f is not detected, (A2) implies that values are read from the same memory addresses in both correct and faulty processors. Since the memory cells where the test program refers to have known values, unknown values are not propagated from the outside of the processor. Moreover, if f is not detected, condition (2) implies that there is no error on control signals or masking circuits of tri-state buses. Therefore, any output of any bus has a value of some activated input of the bus or a known value generated by its masking circuit. Since the value of any RTL signal can be determined by values of primary inputs, registers, and bus outputs, any RTL signal has a known value.

Then we show that the test pattern reaches M or f is detected. We assume that the test pattern of f does not reach the inputs of M. We consider the registers used in order to justify this test pattern in the correct operation of the

processor. In this case, there is a bit b of a register among them such that b has a different value from the correct value. If f is not detected, any RTL signals have known values and the value of b is an error. Since an error is only caused by f of M, a path P through b from an output of M to an input of M exists and the error is propagated on P. Since at least one RTL signal on each cycle is observable from condition (3), at least one RTL signal on P is observable and the fault is detected. Therefore, the test pattern for f reaches the inputs of M or f is detected.

Finally, we show that the test response of M is propagated to an intended primary output or f is detected. The output of M can be observed at a primary output in the fault-free processor. Therefore, there exists a path P such that an error is propagated from M to an primary output. Suppose the fault is not detected. In this case, an error is not propagated to any primary output, and there exists a module M' such that the error is prevented from propagating on P. If M' is not faulty and errors are propagated to M' only through P, the errors are propagated to the outputs of M'. Therefore, (a) M' is faulty that is $M = M'$, or (b) errors are propagated to some inputs of M' which are not on P.

(a) If M' is M, errors are propagated on a cycle, and are observed from condition (3) and therefore f is detected.

(b) If errors are propagated to some inputs of M' which is not on P, errors are propagated on two reconvergent paths from M to M'. By condition (4), errors are observed and f is detected.

Therefore, a fault f detected by MUT test generation can be detected during the execution of a test program synthesized from the test pattern for f.

□

6. Design for Testability Avoiding Error Masking of Software-Based Self-Test

6.1 Formulation

We propose a DFT method to avoid error masking. First, we consider the following DFT element since we add only initialization functions of registers and

observable poins to the original design in order to satisfy the sufficient condition in theorem 1.

- Add a function to initialize a register
- Add an observation point to an RTL signal

Since an advantage of SBST is the possibility of at-speed testing, it is important that the processor after DFT also preserves the possibility of at-speed testing. Therefore, we capture the values of RTL signals at the normal operational speed. We use a multiple input signature register (MISR) for this purpose.

In order to satisfy the sufficient condition, it is necessary to add an initialization function to a register which do not have it. Therefore, it is not necessary to consider an optimization problem which adds an initialization function. We formulate the problem to minimize the number of observation points as follows.
***Error Masking Resolution Problem*:**

Input: An RTL description of a processor

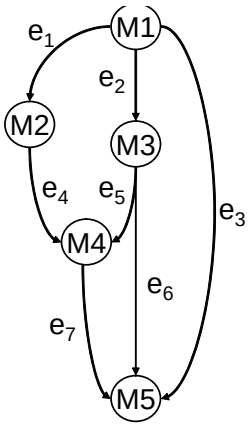
Output: An RTL description of an augmented processor that can achieve 100% template level fault efficiency for any template-based test program

Objective: To minimize the sum of the bitwidths of RTL signals that are made observable

6.2 Algorithm

We propose a heuristic algorithm in order to solve the error masking resolution problem. In the proposed algorithm, we utilize a circuit graph, a reconvergent structure and a reconvergent path dependency graph.

Definition 9 (Circuit Graph) *The circuit graph is a directed graph of an RTL circuit $G_C = (V_{G_C}, E_{G_C})$, where $v \in V$ is a vertex corresponding to a combinational module, a sequential module, a register, a primary input and a primary output and $e \in E_{G_C}$ is an edge corresponding to an RTL signal and has the weight corresponding to the bitwidth of the RTL signal.*



Path:

p_1: e_1, e_4, e_7
p_2: e_2, e_5, e_7
p_3: e_2, e_6
p_4: e_3
p_5: e_1, e_4
p_6: e_2, e_5
p_7: e_5, e_7
p_8: e_6

Figure 4.5. The circuit graph of the reconvergent structure.

Definition 10 (Reconvergent Structure) *Let M and M' be modules. A set of all the paths from M to M' is called a reconvergent structure S.*

Definition 11 (Path Dependency Graph) *Let $V_{ReconvP}$ be a set of paths in all the reconvergent structures. Let $E(p)$ be a set of edges in a path p. A reconvergent path dependency graph is a bipartite graph $G_{RPD} = (V_{ReconvP} \cup V_e, E_{G_{RPD}})$, where $V_e = \bigcup_{p \in V_{ReconvP}} E(p)$, and $E_{G_{RPD}} = \{(p, e) \mid p \in V_{ReconvP}, e \in E(p)\}$.*

Figure 4.5 illustrates an example of circuit graph of a reconvergent structure and names of paths. Figure 4.6 illustrates a path dependency graph corresponding to the reconvergent structure in Figure 4.5. From the path dependency graph, we can identify which paths share an edge.

The proposed algorithm consists of the following four steps.

Step 1: For each register, an initialization function is added if the register does not have the function, and all the control signals of each tri-state bus and all the control signals of its masking circuits are made observable.

Step 2: The circuit graph G_C of the processor is generated.

Step 3: For each cycle in the circuit graph, at least one RTL signal on the cycle is made observable.

Step 4: For each reconvergent path, at least one RTL signal is made observable.

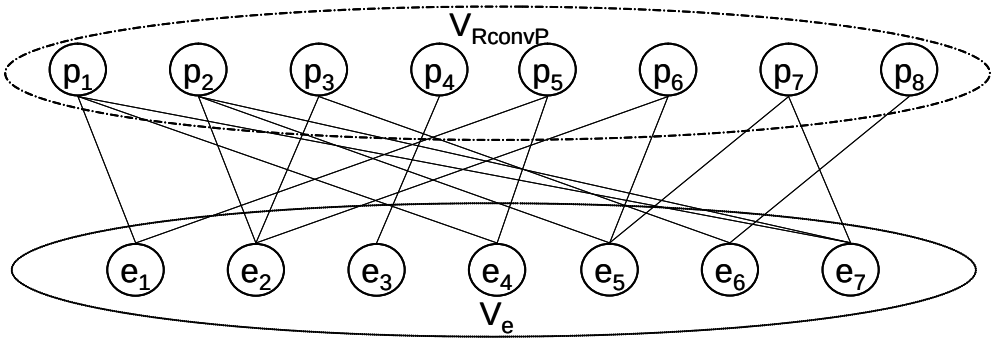


Figure 4.6. The path dependency graph.

We describe the details the Step 1, Step 3 and Step 4 of the algorithm as follows because the circuit graph generated at Step 2 has already defined.

Step 1:

For each register in the processor, an initialization function is added. This initialization function is controllable from a primary input. Since, in general, the processor needs some controls from the outside, a primary input utilized for initializing the registers can be shared for such control. Therefore, it is not necessary to add a new primary input.

Step 3:

In this step, we find a set of RTL signals such that every cycle has at least one RTL signal in the set and the sum of the bitwidths of RTL signals in the set is the minimum. We perform the following four steps.

Step 3.1:

We find a cycle C such that the sum of the weight of edges is the minimum by using the minimum cost to profit ratio cycles algorithm in [18]. Then the edge e_i with the minimum weight in C, e_i is removed from G_C. This process is repeated until G_C becomes acyclic. For later steps, we store the set C_{min} of selected cycles. All the removed edges are restored to G_C.

Step 3.2:

Let E_{cut} denote a set of edges corresponding to the RTL signalsto be observed. We initialize E_{cut} to be empty. From the set of edges in C_{min}, the edge e_i with the minimum weight among the edges that appear in the maximum number of

cycles in C_{min} is selected. The edge e_i is removed from G_C and added to E_{cut}, and the cycles that include e_i are removed from C_{min}. This process is repeated until C_{min} becomes empty.

Step 3.3:

Step 3.1 and 3.2 are repeated until G_C becomes acyclic. The circuit graph obtained in this step is referred to as G_C^{α}.

Step 3.4:

If some of the edges in E_{cut} obtained by processing from Step 3.1 to Step 3.3 is added to G_C^{α}, the circuit graph may not become cyclic. For each $e_c \in E_{cut}$, if the circuit graph becomes acyclic when e_c is added to it, e_c is removed from E_{cut} and e_c is restored to G_C. The circuit graph obtained in this step is referred to as G_C^{β}.

Step 4:

Let P be a set of paths in all the reconvergent structures in G_C^{β}. We find a set of RTL signals such that every path has at least one RTL signals in the set and the sum of the bitwidths of RTL signals in the set is minimum. We perform the following four steps.

Step 4.1:

We generate the path dependency graph $G_{RPD} = (V_{ReconvP} \cup V_e, E_{G_{RPD}})$ from all the reconvergent structures in G_C^{β}.

Step 4.2:

For each vertex $v_i \in V_e$ in G_{RPD}, we calculate a bit rate R_{v_i}. The bit rate R_{v_i} is obtained by $R_{v_i} = \frac{W_{v_i}}{N_{v_i}}$, where W_{v_i} is the bitwidth of the RTL signal corresponding to v_i and N_{v_i} is the outdegree of v_i in G_{RPD}. The bit rate means how may bits needed to make one path observable, and is used to observe more paths by less bitwidths.

Step 4.3:

The edge e_i with the minimum bit rate in $E_{G_{RPD}}$ is selected, and v_i and its neighbors are removed from G_{RPD}. The edge which corresponds to v_i is added to E_{cut}.

Step 4.4:

Steps 4.2 and 4.3 are repeated until the number of paths in each reconvergent structure is less than or equal to one.

Table 4.1. Characteristics of processors.

Processor	#Gate	#Register	#Module	#Instruction
SAYEH	6141	12	10	29
Dlx_N	34032	50	95	25

As the result of processing these steps, we observe RTL signals corresponding to edges in E_{cut}. These RTL signals are connected to an MISR.

7. Experimental Results

We evaluate the proposed method using a non-pipelined processor SAYEH[19] and a five-stage pipelined processor Dlx_N that is based on Dlx processor[20]. Table 4.1 shows the characteristics of SAYEH and Dlx_N. The column titled "#Gate" denotes the number of primitive gates. The column headed "#Register" and "#Module" denote the number of registers and the number of modules at RTL in the processor, respectively. The column "#Instruction" denotes the number of instructions defined in an instruction set architecture (ISA). The numbers of gates in SAYEH and Dlx_N processor are 6,141 and 34,032, respectively. The numbers of registers and modules at RTL in both processors are 12 and 50, and 10 and 95, respectively. Both processors have the standard 29 and 25 instructions, respectively. All the registers of SAYEH processor are resetable. All the registers except for registers of the register-file of Dlx_N processor are resetable.

Table 4.2 shows hardware overhead of the proposed method and the full-scan design for SAYEH and Dlx_N. The column "DFT" denotes the design for testability method applied to both processors, where "FS" and "PM" denote the full-scan design method and the proposed method, respectively. The columns "Area" and "HO" denote the area of the processor and the hardware overhead, respectively. In the columns "Area," "Original" and "Additional" denote the original area of the processor without DFT and the additional area that increases by applying the proposed DFT method to the processor, respectively. A unit of the area sets a not gate to one. In the columns "Additional" and "HO", the sub-

columns “Init. Func.” and “OB” denote the additional area and the hardware overhead of the initialization function and MISR that increases by applying the proposed DFT method to the processor, respectively. In these sub-columns, “-” denotes no additional area and no hardware overhead. The column “#OB” denotes the number of observable bits. In the full-scan design method and the proposed method, an observable bit means the number of scan flip-flops and the number of inputs of MISR. There are not any additional area or hardware overhead of the initialization function since all the registers of SAYEH processor are resetable. In the case of the full-scan design method, both processors do not induce the additional area and the hardware overhead of the initialization function since FFs are only modified into SFFs. The number of observable bits of the proposed method becomes less than that of the full-scan design method for both processors. For Dlx_N processor, the hardware overhead of the proposed method is smaller than that of the full-scan design method. The Dlx_N processor has many registers including the architecture registers that appear in instruction set architecture and the pipeline registers to enhance the performance. Therefore, the full-scan design induces a large area overhead. For details of the area for Dlx_N processor, the area of the initialization function is almost the same as MISR for observable points. However, if Dlx_N processor has already had the initialization function for all the registers, the area of the initialization function is not required. The area of the initialization function depends on the design specification of the processor. On the other hand, for the SAYEH processor, the hardware overhead of the proposed method is larger than that of the full-scan design method. This is because that the SAYEH processor has a very area-optimized design with a lot of loops and a few registers; therefore, the proposed method needs many observation points whereas full-scan design requires little area overhead. Moreover, the hardware overhead per one observed bit of the proposed method is larger than for the full-scan design. However, this hardware overhead can be reduced if we compress the observed space before applying it to MISR.

In order to show the effectiveness of the proposed method, we apply the proposed DFT method to the arithmetic logic unit (ALU) in Dlx_N processor. Table 4.3 and Table 4.4 show the results of the MUT test generation for the

Table 4.2. Hardware overhead.

Processor	DFT	Area			#OB	HO(%)	
		Original	Additional				
			Init. Func.	OB		Init. Func.	OB
SAYEH	FS	12389	-	1485	165	-	11.99
	PM		-	2958	102	-	23.88
DLX_N	FS	55995	-	13635	1379	-	23.23
	PM		3968	4379	151	7.09	7.46

Table 4.3. MUT test generation for ALU.

Total	RF	DF	FC	FE	TGT (sec)
7030	12	7018	99.83	100.00	358.70

ALU in Dlx_N processor and the execution of the template-based test program before/after the proposed DFT method is applied, respectively. We used the SBST method in [16] as an MUT test generation and a test program generation.

In Table 4.3, the columns "Total", "RF", "DF", "FC", "FE" and "TGT" denote the number of total faults of ALU, the number of the identified redundant faults, the number of the detected faults, the fault coverage, the fault efficiency, and the total test generation time for the MUT test generation, respectively. A unit of "TGT" is second. In order to identify redundant faults, we use the method in [16]. The total fault coverage and fault efficiency are 99.83% and 100.00%, respectively. The total test generation time is 358.70 second. This test generation time is reasonable because a combinational test generation is applied to each constraint circuit synthesized by the method in [16] which is a combinational circuit.

In Table 4.4, the columns "DF", "EM", "FC", "FE", "FE_T" and "TAT" denote the number of the detected faults, the number of faults undetected by error masking, the fault coverage, the fault efficiency, the template level fault efficiency and the test application time during the execution of the test program,

Table 4.4. Test program execution for ALU.

DFT	DF	EM	FC	FE	FE_T	TAT (clock)
Before	6948	54	98.83	99.00	99.26	7508
After	7022	0	99.89	100.00	100.00	7124

respectively. A unit of "TAT" is clock. In Table 4.4, there exist 54 faults undetected by error masking before the proposed DFT method. However, after the DFT method is applied, the number of faults undetected by error masking is 0. The proposed DFT method can achieve 100% template level fault efficiency. The fault coverage after the proposed DFT is larger than that of before the DFT. Moreover, the proposed DFT method can also reduce about 5% of the total test application time.

8. Summary

In this chapter, we showed a sufficient condition to avoid error masking for template-based test programs, and proposed a design for testability method to satisfy the sufficient condition. The experimental results reveal that the proposed method achieves less hardware overhead than full-scan design if the processor features many registers and less loops or reconvergent paths. In general, modern processors oriented to high performance have many registers to accelerate their speed, while the structure tends to be simpler than the design oriented to area optimization. From this observation, we consider that the proposed method is suitable for such modern processors. Since the proposed method adds only observation points to the original design, it enables at-speed testing. The reduction of the hardware overhead caused by the DFT method is the issue to be investigated in our future work.

Chapter 5

Conclusions and Future Work

When LSIs with high performance and a lot of functions are produced by VDSM technologies, LSIs testing requires at-speed testing and short test generation time. In order to meet these requirements, this thesis presented a test generation method for an LSI circuit and a design for testability (DFT) method which is helpful to software-based self-test for processors.

In Chapter 3, we propose a method of accelerating test generation for sequential circuits using the knowledge about a set of state justification sequences, the bound on the maximum length of state distinguishing sequences, the information about the valid states and the value of the reset state. We assume that circuits are given in FSM description. For circuits designed at register transfer level (RTL), controllers of the circuits are generally specified by FSM description. The proposed method is effective for such controllers. The sequential circuit is synthesized from a given FSM by a synthesis for testability (SFT) method proposed in this chapter which takes the features of our test generation method into consideration. The SFT method guarantees the existence of state distinguishing sequences of the specified length by making the given FSM reduced. Thus, the performance of the test generator is improved as it uses state justification sequences extracted from the completely specified state transition function of the FSM produced by the synthesizer.

In Chapter 4, we proposed a design for testability method of software-based self-test for processors. We proved that the proposed method can achive 100% template level fault efficiency if a processor satisfies three conditions: (1) each

register is initialized at the beginning of the execution of the test program, (2) all the control signals of each tri-state bus and its masking circuits are observable, (3) for each cycle, at least one RTL signal on the cycle is observable, (4) for each pair of reconvergent paths, at least one RTL signal on the two paths is observable. The experimental results reveal that the proposed method achieves less hardware overhead than full-scan design if the processor features many registers and less loops or reconvergent paths. From this observation, we consider that the proposed method is suitable for such modern processors. Moreover, in the experimental results, we showed that the proposed design for testability resolved error masking and reduced the test application time of the ALU in Dlx_N processor. Since the proposed method adds only observation points to the original design, it enables at-speed testing.

Finally, we discuss our future work. In Chapter 3, the test generation time of the proposed test generation method is faster than that of the commercial test generation. However, the area of the sequential circuit synthesized by the proposed synthesis for testability (SFT) method may be larger than the area of the sequential circuit synthesized by an ordinary synthesizer. In experimental results, the area of the benchmark circuit "pma" is especially large. In order to reduce hardware overhead, it leaves some room for consideration, where it is how to assign coordinates with don't cares in input vectors and output vectors of the FSM during SFT. In Chapter 4, MISR is utilized for observing RTL signals. If an error propagating to an observable RTL signal does not depend on propagating to other observable RTL signals, we can compact the observable RTL signal with other observable RTL signals. Therefore, we have possibility of further reducing the hardware overhead compared with the proposed design for testabilty method. We should investigate how to reduce hardware overhead of each method and the algorithm for each method in future.

Acknowledgements

Many people have supported me during my Ph.D. studies [1] .

I would like to acknowledge their kind support.

I am very much grateful to my supervisor Professor Hideo Fujiwara for his gracious guidance and advice.

I would like to thank Professor Hiroyuki Seki of NAIST (Nara Institute of Science and Technology) for his valuable comments.

I also would like to express my gratitude to Associate Professor Michiko Inoue of our laboratory for her frequent, stimulating and helpful discussions.

I am deeply indebted to Assistant Professor Satoshi Ohtake of our laboratory for his frequent, stimulating and helpful discussions.

I also wish to thank Assistant Professor Tomokazu Yoneda of our laboratory for his friendly discussions and encouragement.

I would like to thank Professor Kewal K. Saluja, University of Wisconsin-Madison, USA, for friendly discussions and useful comments.

I am grateful to Professor Tomoo Inoue of Hiroshima City University for his friendly helpful advice for submitting my journal paper and giving me useful comments.

I would like to thank Mr. Masahide Miyazaki of STARC (Semiconductor Technology Academic Research Center) and Mr. Yasuyuki Nozuyama of Toshiba Corporation (Semiconductor Company) for their useful comments.

My thanks also go to the present and former members of our laboratory.

Finally, I wish to thank my parents and my little sister for their continuing support and encouragement.

[1] This work was supported in part by 21st Century COE (Center of Excellence) Program "Ubiquitous Networked Media Computing."

References

[1] H. Fujiwara, "Logic testing and design for testability," The MIT press, Cambridge, 1985.

[2] T. Niermann and J. Patel, "HITEC: A test generation package for sequential circuits," *Proc. of the European Design Automation Conference*, pp. 214–218, 1991.

[3] H. Cho, G. D. Hachtel and F. Somenzi, "Redundancy identification/removal and test generation for sequential circuits using implicit state enumeration," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 935–945, July 1993.

[4] M. Samiha and Z. Yervant, "Principles of testing electronic systems," Wiley-Interscience, 2000.

[5] H. K. Ma, S. Devadas, A. R. Newton and A. Sangiovanni–vincentelli, "Test generation for sequential circuit," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 1081–1093, October 1988.

[6] T. P. Kelsey and K. K. Saluja, "Fast test generation for sequential circuits," *Int. Conf. on Computer-Aided Design 1989*, pp. 354–357, November 1989.

[7] J. Gross and J. Yellen , *Graph theory and its applications*, CRC press, 1991.

[8] C. W. Cha, W. E. Donath and F. Özgüner, "9-v algorithm for test pattern generation of combinational digital circuits," *IEEE Trans. Computers*, vol. C–27, pp. 193–200, March 1978.

[9] P. Muth, "A nine-valued circuit model for test generation," *IEEE Trans. Computers*, vol. C–25, pp. 630–636, June 1976.

[10] The CAD benchmarks at North Carolina State University, http://www.cbl.ncsu.edu/www/.

[11] W. -C. Lai, A. Krtic and K. -T. Cheng, "Test program synthesis for path delay faults in microprocessor cores," *Proc. of International Test Conference 2000*, pp. 1080-1089, 2000.

[12] W. -C. Lai, A. Krtic and K. -T. Cheng, "Instruction-Level DFT for testing processor and IP cores in system-on-a chip," *Proc. of Design Automation Conference 2001*, pp. 59-64, 2001.

[13] L. Chen and S. Dey, "Software-based self-testing methodology for processor cores," *IEEE Trans. on CAD*, vol. 20, no. 3, pp. 369-380, 2001.

[14] L. Chen, S. Rabi, A. Raghunath and S. Dey, "A scalable software-based self-test methodology for programmable processors," *Proc. of Design Automation Conference 2003*, pp. 548-553, 2003.

[15] K. Kambe, M. Inoue and H. Fujiwara, "Efficient template generation for instruction-based self-test of processor cores, " *Proc. of IEEE 13th Asian Test Symposium (ATS'04)*, pp. 152-157, 2004.

[16] M. Inoue, M. Nakazato, S. Yokoyama, K. Kambe and H. Fujiwara, "Efficient and Effective Test Program Generation for Software-Based Self-Test of Pipelined Processors, " *NAIST Technical Reports*, No.2006005, Aug. 2006.

[17] M. Inoue, K. Kambe, N. Hoashi, and H. Fujiwara, "Instruction-based self-test for sequential modules in processors," *Proc. of IEEE 5th Workshop on RTL and High Level Testing (WRTLT'04)*, pp. 109-114, 2004.

[18] M. Näher, "LEDA," Cambridge university press, 1999.

[19] Z. Navabi, "VHDL analysis and modeling of digital systems," McGraw-Hill, 1997.

[20] J. H. Hennesy and D. A . Patterson, "Computer Architecture: A quantative approach," Morgan Kaufmann Publishers, 1996.

[21] Semiconductor Technology Academic Research Center (STARC), "RTL design style guide," STARC, 2000. (In Japanese).

Appendix

A. Dlx_N Processor

I designed Dlx_N processor to evaluate the proposed design for testability method. Dlx_N is a 32 bit, 5-stage pipelined RISC processor. It has 26 most common instructions. It consists of 32 general purpose 32 bit registers.

Instruction Set

1. NOP	6. SUBU	11. ANDI	16. SW	21. SLTI
2. ADD	7. ADDIU	12. ORI	17. LUI	22. SLTU
3. SUB	8. MFC0	13. SLL	18. BEQ	23. SLTIU
4. ADDI	9. AND	14. SRL	19. BNE	24. J
5. ADDU	10. OR	15. LW	20. SLT	25. JR
				26. JAL

Instruction Set Architecture

1. Register-Register Type Instruction (R-Type)

OP	RS	RT	RD	SHAMT	FUNCT
(6 bit)	(5 bit)	(5 bit)	(5 bit)	(5 bit)	(6 bit)

2. Immediate Instruction (I-Type)

OP	RS	RT	IMMEDIATE
(6 bit)	(5 bit)	(5 bit)	(16 bit)

3. Jump Instruction (J-Type)

OP	TARGET ADDRESS
(6 bit)	(26 bit)

OP: This field corresponds to an operation code distinguishing all the operations except for R-Type instructions.

RS: This field corresponds to the address of a source register.

RT: This field usually corresponds to the address of a source register in case of R-Type instructions. However, this field corresponds to the address of a destination register in case of I-Type instructions.

RD: This field corresponds to the address of a destination register in case of R-Type instructions.

SHAMT: This field corresponds to a shift amount in case of R-Type instructions.

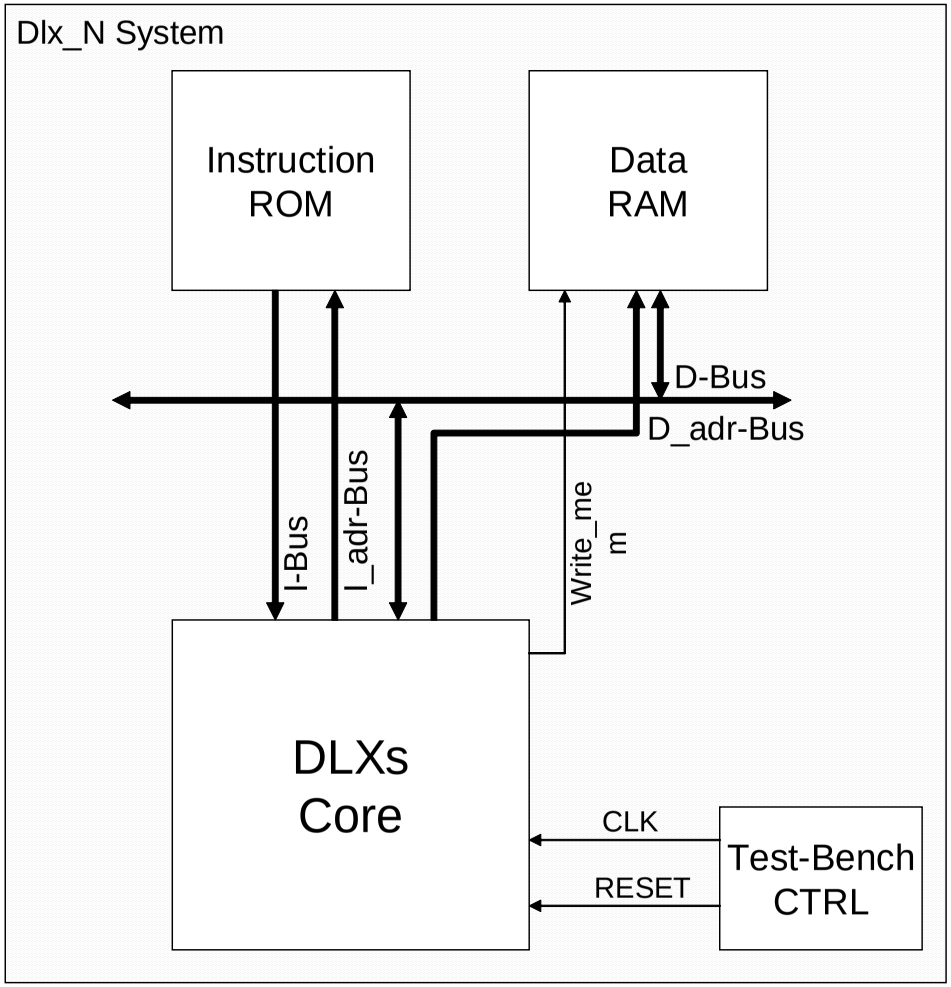
IMMEDIATE: This field corresponds to an immediate value in case of I-Type instructions.

Instruction Encoding

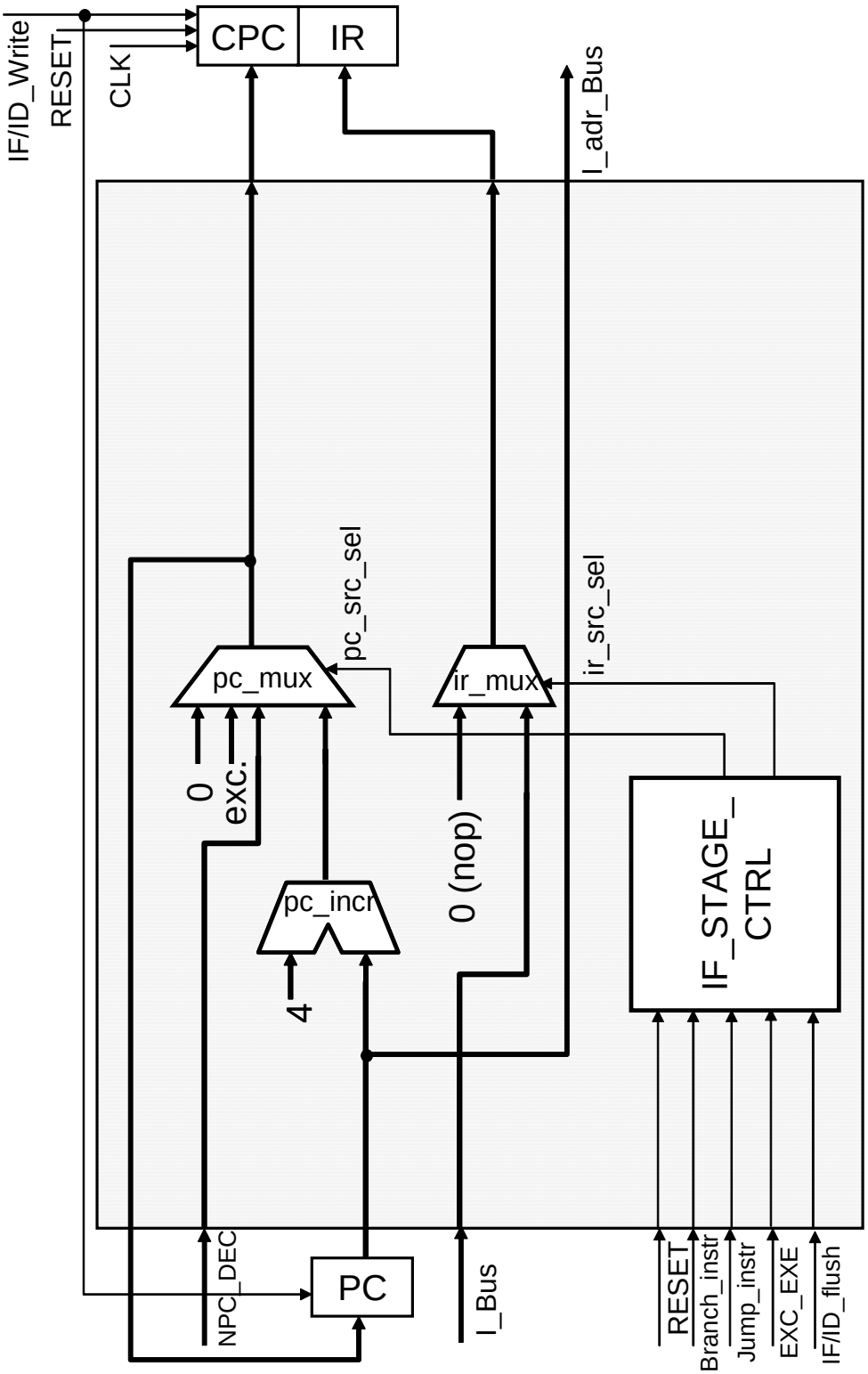
Instruction	Operation	OP	FUNCT	Type
NOP	No operation	000000	000000	R
ADD RD, RT, RS	RD $\Leftarrow$ RS + RD (Overflow)	000000	100000	R
SUB RD, RT, RS	RD $\Leftarrow$ RS - RT (Overflow)	000000	100010	R
ADDI RT, RS, IMMEDIATE	RT $\Leftarrow$ RS + IMMEDIATE (Overflow)	001000	-	I
ADDU RD, RT, RS	RD $\Leftarrow$ RS + RT	000000	100001	R
SUBU RD, RT, RS	RD $\Leftarrow$ RS - RT	000000	100011	R
ADDIU RT, RS, IMMEDIATE	RT $\Leftarrow$ RS + IMMEDIATE	001001	-	I
MFC0 RD, RT, RS	RD $\Leftarrow$ $EPC	010000	000000	R
AND RD, RT, RS	RD $\Leftarrow$ RS and RT	000000	100100	R
OR RD, RT, RS	RD $\Leftarrow$ RS or RT	000000	100101	R
ANDI RT, RS, IMMEDIATE	RT $\Leftarrow$ RS and IMMEDIATE	001100	-	I
ORI RT, RS, IMMEDIATE	RT $\Leftarrow$ RS or IMMEDIATE	001101	-	I
SLL RD, RT, SHAMT	RD $\Leftarrow$ RT $\ll$ SHAMT	000000	000011	R
SRL RD, RT, SHAMT	RD $\Leftarrow$ RT $\gg$ SHAMT	000000	000010	R
LW RT, RS(IMMEDIATE)	RT = MEM[RS + IMMEDIATE]	100011	-	I
SW RT, RS(IMMEDIATE)	MEM[RS + IMMEDIATE] = RT	001001	-	I
LUI RT, IMMEDIATE	RT $\Leftarrow$ IMMEDIATE	000001	-	I
BEQ RT, RS, IMMEDIATE	if(RT == RS) go to $PC + 4 + IMMEDIATE	000100	-	I
BNE RT, RS, IMMEDIATE	if(RT $\neq$ RS) go to $PC + 4 + IMMEDIATE	000101	-	I
SLT RD, RT, RS	RD $\Leftarrow$ 1 if(RT < RS) else 0 (Overflow)	000000	101010	R
SLTI RT, RS, IMMEDIATE	RT $\Leftarrow$ 1 if(RS < IMMEDIATE) else 0 (Overflow)	001010	-	I
SLTU RD, RT, RS	RD $\Leftarrow$ 1 if(R T< RS) else 0	000000	101011	R
SLTIU RT, RS, IMMEDIATE	RT $\Leftarrow$ 1 if(RS < IMMEDIATE) else 0	001011	-	I
J TARGET_ADDRESS	go to TARGET_ADDRESS	000010	-	J
JR $REG[31]	go to $REG[31]	000000	001000	R
JAL TARGET_ADDRESS	$REG[31] = $PC + 4; go to TARGET_ADDRESS	000011	-	J

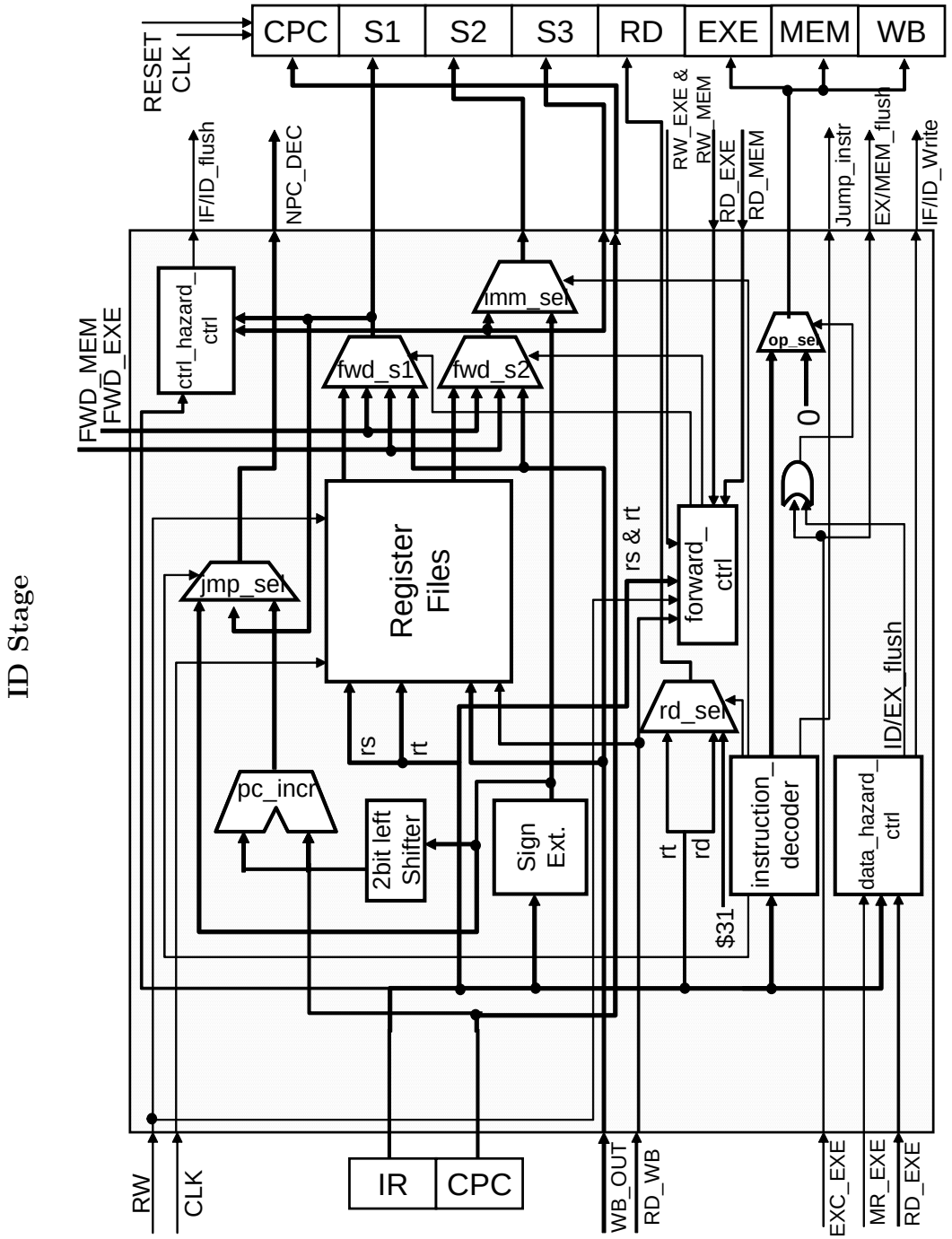
$PC: Program Counter, $EPC: Exception Program Counter

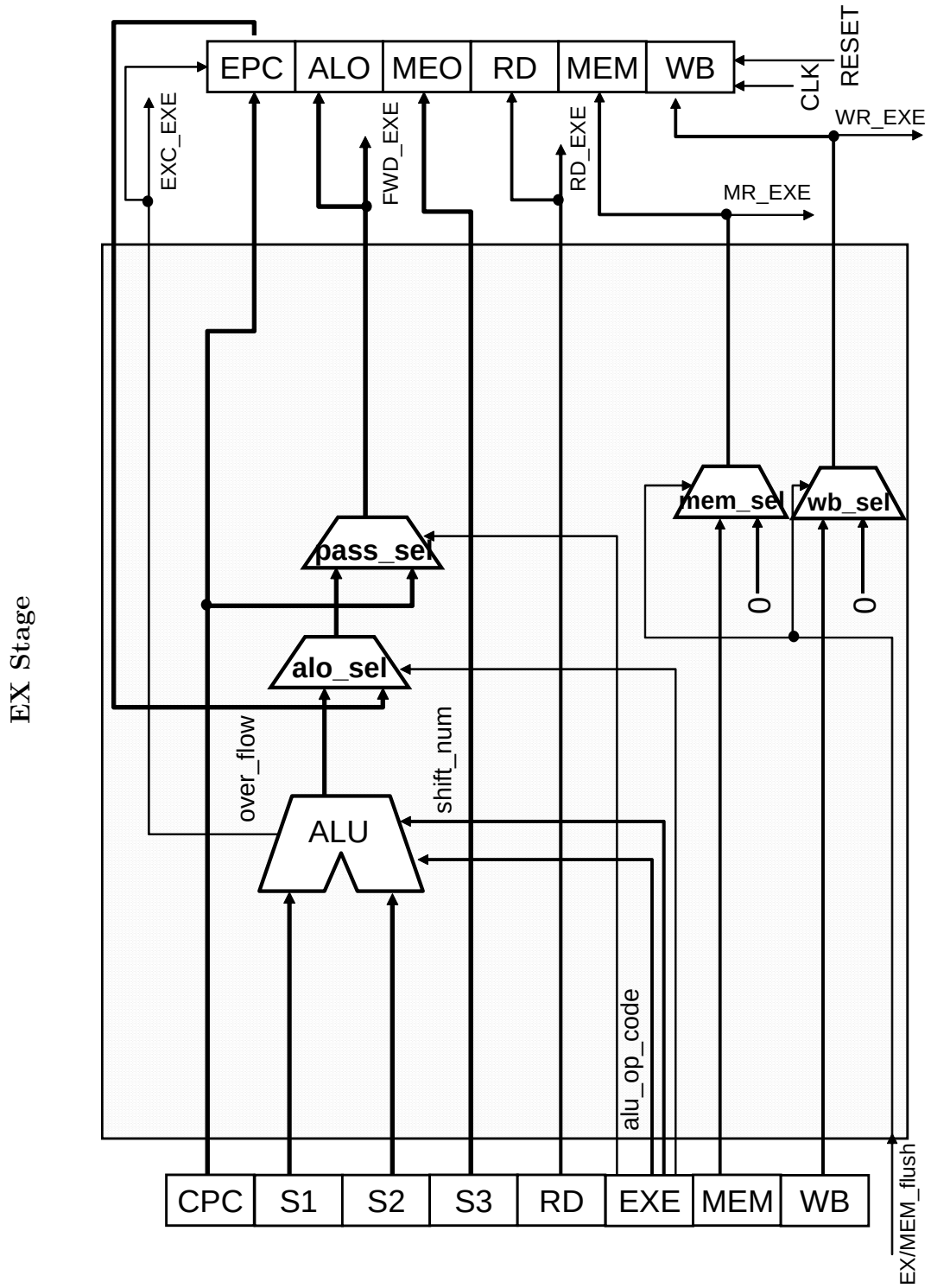
Dlx_N Testbench System

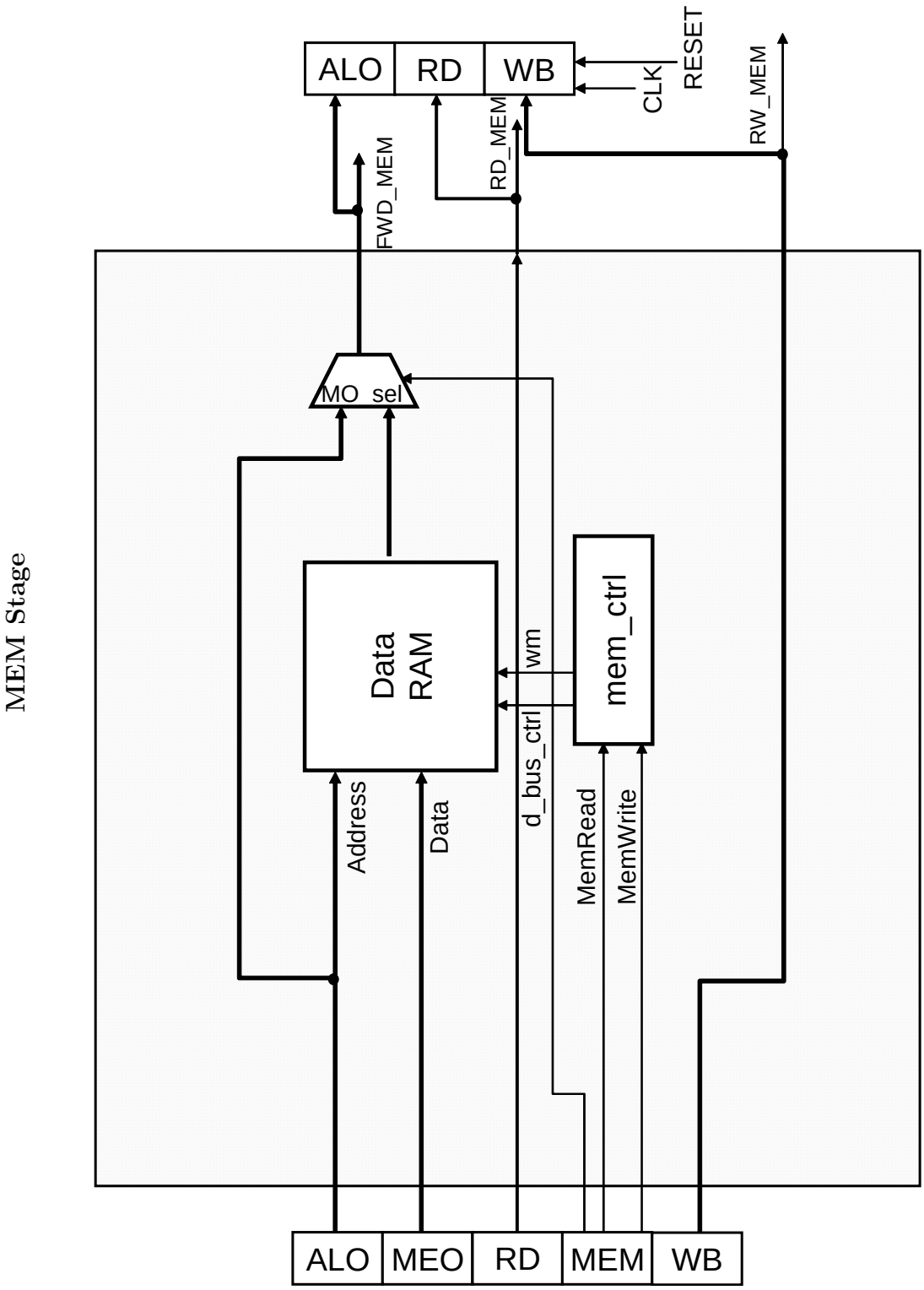


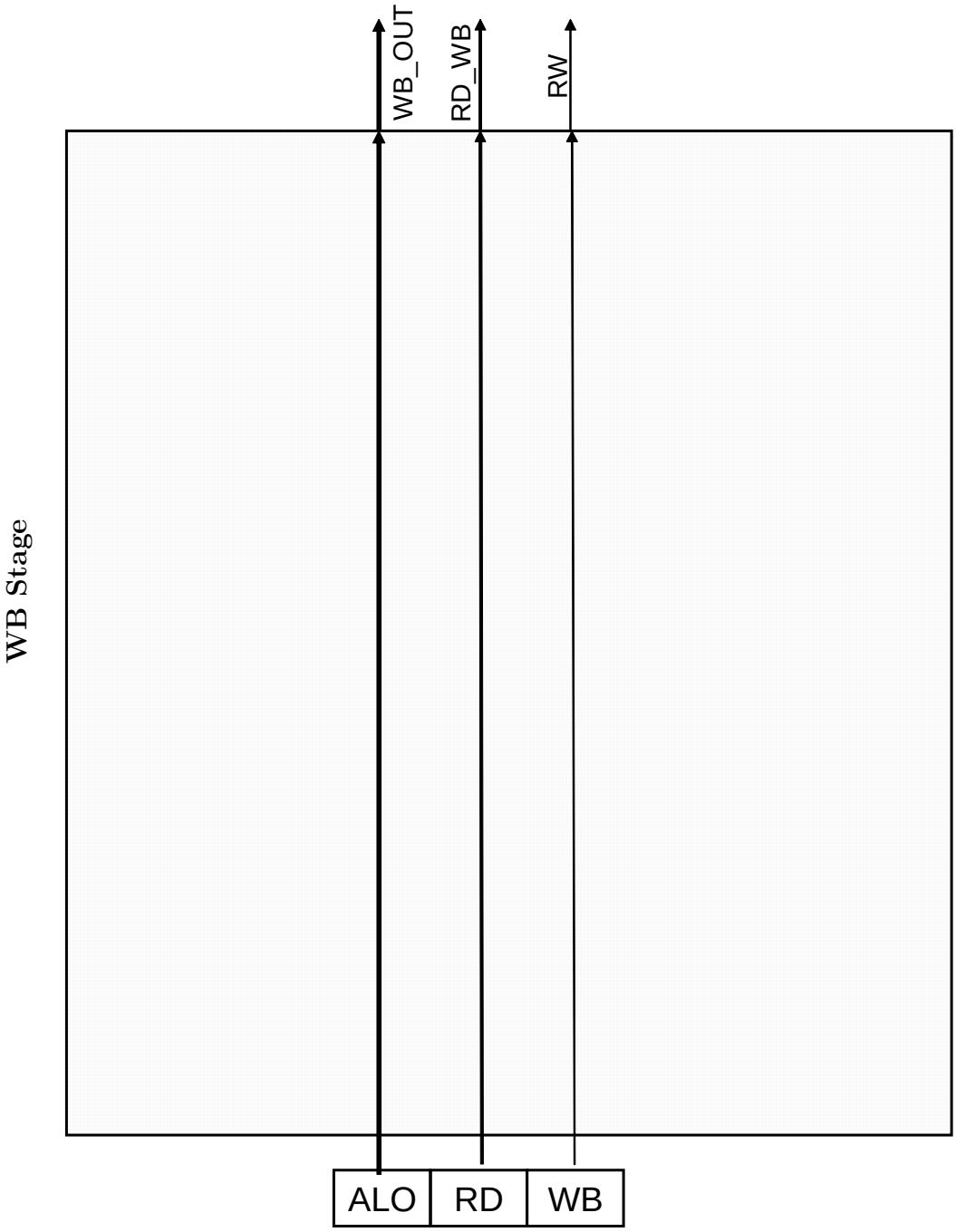
IF Stage


IF/ID_Write
RESET
CLK
CPC
IR
I_adr_Bus
pc_src_sel
ir_src_sel
pc_mux
ir_mux
0
exc.
pc_incr
4
0 (nop)
IF_STAGE_
CTRL
NPC_DEC
PC
I_Bus
RESET
Branch_instr
Jump_instr
EXC_EXE
IF/ID_flush


ID Stage
CPC
S1
S2
S3
RD
EXE
MEM
WB
RESET
CLK
IF/ID_flush
NPC_DEC
RW_EXE & RW_MEM
RD_EXE
RD_MEM
Jump_instr
EX/MEM_flush
IF/ID_Write
FWD_MEM
FWD_EXE
ctrl_hazard_ ctrl
imm_sel
fwd_s1
fwd_s2
op_sel
0
jmp_sel
Register Files
rs & rt
forward_ ctrl
rd_sel
ID/EX_flush
rs
rt
pc_incr
2bit left Shifter
Sign Ext.
rt
rd
instruction_ decoder
data_hazard_ ctrl
$31
RW
CLK
IR
CPC
WB_OUT
RD_WB
EXC_EXE
MR_EXE
RD_EXE


EX Stage
EPC
ALO
MEO
RD
MEM
WB
CLK
RESET
WR_EXE
MR_EXE
EXC_EXE
FWD_EXE
RD_EXE
mem_sel
wb_sel
pass_sel
alo_sel
0
0
over_flow
shift_num
ALU
alu_op_code
CPC
S1
S2
S3
RD
EXE
MEM
WB
EX/MEM_flush


MEM Stage
ALO
RD
WB
CLK
RESET
RW_MEM
RD_MEM
FWD_MEM
MO_sel
Data RAM
mem_ctrl
wm
d_bus_ctrl
Address
Data
MemRead
MemWrite
ALO
MEO
RD
MEM
WB


WB_OUT
RD_WB
RW
WB Stage
ALO
RD
WB