High-Level Test Synthesis for Delay Fault Testability

Sying-Jyan Wang and Tung-Hua Yeh
Department of Computer Science
National Chung-Hsing University
Taichung 402, Taiwan, ROC

Abstract

A high-level test synthesis (HLTS) method targeted for delay fault testability is presented. The proposed method, when combined with hierarchical test pattern generation for embedded modules, guarantees 100% delay test coverage for detectable faults in modules. A study on the delay testability problem in behavior level shows that low delay fault coverage is usually attributed to the fact that two-pattern test for delay testing cannot be delivered to modules under test in consecutive cycles. To solve the problem, we propose an HLTS method that ensures valid test pairs can be sent to each module through synthesized circuit hierarchy. Experimental results show that this method achieves 100% fault coverage for transition faults in functional units, while the fault coverage in circuits synthesized by LEA-based allocation algorithm is rather poor. The area overhead due to this method ranges from 2% to 10% for 16-bit datapaths.

1. Introduction

Decreasing feature sizes and increasing clock speeds have combined to alter the defect effects dramatically. Recent evidence indicates that delay-inducing defects can no longer be ignored nor go untested [1]. For circuits designed with 130nm or more advanced technologies, the transition fault is considered essential to achieve the acceptable defect level. The detection of delay fault requires at-speed test techniques, which create signal transitions to be captured at normal speed. In order to test delay faults, we need two test patterns, known also as pattern pair. The first test pattern of the pattern pair is initialization pattern (V_1), and the second test pattern is activation pattern (V_2). V_1 is to set up the initial logic value to a known state of the target net and V_2, combined with V_1, is to activate a required transition to test a target fault.

Scan-based test techniques [2],[3] are capable of at-speed testing. However, there are many complicating factors when moving from relatively slow scan-based tests for stuck-at faults to testing for delay faults. At-speed testing cab be carried out through functional patterns, but it may be difficult to achieve good delay fault coverage. In order to conduct delay testing in the functional path, one needs to provide a pair of independent patterns to the module under test in two consecutive cycles, which may not be possible in normal operations.

The behavioral synthesis refers to the process of producing an RTL circuit from a behavioral description [4]. The behavioral description of a circuit is usually specified in a hardware description language like VHDL or Verilog, and it is complied into an intermediate representation called control data flow graph (CDFG). The scheduling process determines the cycle-by-cycle behavior of the design by assigning operations to specific clock cycles or control steps. The allocation process maps operations in a scheduled DFG (SDFG) to modules, assigns variables to registers and constructs the interconnection structure among modules and registers by using multiplexers or tri-stated buses. The controller is then generated according as the cycle-by-cycle behavior of circuits to provide the required sequence of control signals to select paths in multiplexers and load data into registers.

The testability in high-level synthesis were discussed in [5],[6], and a similar research was proposed in [7]. The main advantage of such methods is that they reduce test generation time significantly. However, since sequential automatic test pattern generation (ATPG) is necessary, it is difficult to apply this method to large circuits. A method combining hierarchical test generation [8] and behavioral synthesis is proposed in [9]. In this work, the authors tried to derive a control path from primary inputs to an operation's input and an observation path from the operation's output to primary outputs from a given CDFG. If such two paths exist, they are referred to as the test environment, and the module that carries out this operation is testable since the pre-computed test patterns are known. If the test environment does not exist, extra test multiplexers are added so that the test environment can be constructed. The advantage of this method is that only combinational ATPG is needed, and thus it is suitable for large designs. Many similar methods were proposed for RTL designs [10],[11]. Most of the high-level test synthesis (HLTS) methods only consider stuck-at faults.

The delay fault testability in high-level design are considered in two recent works [12],[13]. These methods improves delay fault testability by adding DFT features, such as test multiplexers, through functions and hold functions, to circuit under test. Since these methods do not exploit the behavioral information given in the CDFG, the area overhead can be very large. Another problem is that test control signals must be provided externally; that is, special test architecture, such as the enhanced scan chain, is needed to provide the test control signals, which not only includes high area overhead but also requires complicated timing signals in the test process.

In our paper, we present a novel integrated method with hierarchical test generation and behavioral synthesis for delay fault testability. The proposed method is more flexible since it starts at the behavioral level, and thus it exploits behavioral information provided in CDFG to achieve better delay fault coverage with lower hardware penalty. Furthermore, the testing is conducted in normal mode and thus there is no need for special test architecture.

2. Preliminaries

Delay testing verifies the whether a circuit is functionally correct under their operating frequency. Many different delay fault models have been developed, in which the transition fault model is more frequently used in recent years due to its simplicity and ease of ATPG. In this paper, we present our method and show the experimental results according to the transition fault model. However, the proposed method itself is independent of the fault model used and is applicable to any type of delay testing.

2.1 Transition Fault Testing

In the transition fault model, delay defects are modeled as two types of faults: slow-to-rise (STR) fault and slow-to-fall (STF) fault. We need to apply a pair of test patterns $<V_1, V_2>$ for the detection of transition faults. An STR (STF) transition fault can be detected if and only if following two conditions are satisfied.

1. The initialization pattern, V_1, sets the target line to 0 (1).
2. The launch pattern, V_2, can make a rising (falling) transition and V_2 is a sa-0 (1) pattern.

2.2 Delay Fault Testability: A Behavioral View

The low delay fault coverage is usually caused by the circuit architecture. The testability problems can be attributed to: (1) high data dependency in the behavioral description, and (2) no path existing from primary inputs to inputs of an RTL module or from output of an RTL module to primary outputs (e.g. self loop or constant multiplication).

The reason that high data dependency may reduce delay fault coverage can be explained by the example shown in Figure 1.

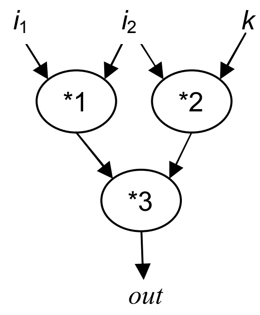

Figure 1 CDFG

For the CDFG shown in Figure 1, i_1 and i_2 are input variables applied to the primary inputs, *out* is an output variable. Assume that operations *1 and *3 are allocated to the same multiplier unit M in two consecutive cycles, and a pre-computed test pair $<V_1, V_2>$ for M is known. If we want to test module M in the normal functional mode, we need to deliver V_1 and V_2 from the primary inputs to the inputs of module M and to propagate M's output response to the primary output. As long as all pre-computed test pairs of a multiplier can be applied to module M through the circuit architecture, M is delay fault testable. Let V_{1l} (V_{2l}) denotes the part of V_1 (V_2) that is applied to the left input port of M, while V_{1r} (V_{2r}) is the part of V_1 (V_2) applied to the right input port of M. The hierarchical test patterns can be assigned according to the CDFG as follows.

$$V_{1l} = i_1 \quad (1)$$
$$V_{1r} = i_2 \quad (2)$$
$$V_{2l} = i_1 * i_2 \quad (3)$$
$$V_{2r} = i_2 * k \quad (4)$$

If each pattern pair $<V_1, V_2>$ satisfies the above architectural constraints, module M is delay fault testable. However, in most cases it may not be possible to derive such test pairs due to data dependency in the CDFG. For example, let a test pair for the multiplier be $V_{1l} = 10$, $V_{1r} = 30$, and $V_{2l} = 20$. However, since $i_1 * i_2 = 300$, Eq. (3) is violated, which means the test pair $<V_1, V_2>$ cannot be applied to module M under normal operation. Unfortunately, the data dependency appears in many CDFGs, especially the DSP algorithms. Therefore, it is difficult to achieve high delay fault coverage if we directly synthesize a given CDFG without considering testability issues.

Another problem of the DFG is that one of the two inputs of *2 is a constant k. As a result, the multiplier module executing this operation cannot be applied with most pre-computed patterns in the normal-mode operation. In general, it is difficult to derive hierarchical test patterns for operations with constant operand.

2.3 Behavioral Synthesis for Delay Fault

Since the target circuits are non-scan RTL architectures, we must be able to apply two independent test patterns to the input ports of an RTL module through the circuit hierarchy in two consecutive clock cycles to make the module delay fault testable. Also the output response to the second pattern must be captured and propagated to one of the primary outputs. The testability of a module is mainly determined by the overall RTL architecture, which is the result of scheduling and allocation process in the high-level synthesis. The following terminology will be used in the proposed synthesis procedure.

Definition 1: If it is necessary to store the result of an operation into a register, the operation is a *real operation*; otherwise it is a *spurious operation*.

In the original CDFG all operations are real operations. However, a lot of spurious operations will be created in the synthesized circuit even though they are not defined in the original CDFG.

Definition 2: An operation in the CDFG is a *controllable operation (CO)* if its input pattern satisfies the following conditions.

1. The input pattern can be applied directly from one or more primary input variables, or delivered indirectly through other controllable operations.
2. If the operation has at least two inputs, they must be individually controllable. In other words, their input cones are disjoint.

If an operation is a controllable, the set of primary input variables belonging to its input cone are called *controllable primary input variable set (CPIVS).*

For example, operation *1 in Figure 1 is a CO, while operation *3 is not.

Definition 3: Consider two operations o_1 and o_2, and let CPIVS_1 and CPIVS_2 be their respective input set. The two operations form a *test operation pair (TOP)* if they satisfy the following four conditions.

1. The two operations are active in two consecutive cycles.
2. They are the same type of operation.
3. Both operations are COs, and $\mathrm{CPIVS}_1 \cap \mathrm{CPIVS}_2 = \varnothing$ when the variables are alive.
4. The second operation o_2 must be a real operation and its output variable is assigned to one of the primary output registers.

An operation satisfies conditions 1 and 2 is capable of carrying out a two-pattern test, and condition 3 guarantees that the required two patterns can be derived through their associating CPIVS without affecting the original circuit behavior. Condition 4 ensures the faulty effect can be observed directly through a primary output register. While such TOP is assigned to the same module, this module will be two-pattern testable.

3. Proposed Method

As we discussed in Section 2, the major sources causing low delay fault coverage in the datapath components can be attributed to data dependency and operations with constant inputs. It may not be possible to find valid test operation pairs in the original CDFG due to these two reasons. Fortunately, in most cases not all the functional modules are busy with real operations simultaneously. Therefore, if a module is not delay fault testable in the normal mode, it may be possible to introduce some spurious operations that make the module testable. These spurious operations can be enforced by adding some dummy test operations into the original CDFG. The important issue, however, is to find a way to add dummy operations that neither affect normal circuit behavior nor introduce significant area penalty.

3.1 Test Operation

Adding extra test operations will not increase the number of functional units, since they are allocated to existing functional units as well. However, they do introduce requires extra hardware, including multiplexers and registers. Usually it is not possible to eliminate the extra multiplexers since we need to provide additional paths in order to send test patterns to modules under test. Sometimes we need to reload new patterns into registers for test purpose, and this may prolong the lifetime of a variable, which may require extra registers. The overhead due to extra registers, however, can be reduced by selecting test input carefully.

3.1.1 CPIVS Selection. The required primary input variables are loaded in the first control step and their lifetimes may last several cycles. Assuming that a test operation is added in control step L and the lifetimes of some primary input variables are longer than L. In this case, selecting such variables for the added test operation does not prolong the input variables' lifetimes since these variables are still alive.

On the other hand, if a dummy test operation's pattern cannot be derived from the primary input variables applied in the first control step, it is better to select a primary input variable that is dead when this test operation is activated. With this restriction, the dummy test operation will not affect original circuit behavior since the input variable is no longer required, and thus we can load a new pattern to the input variable that is used in the test operation. Note that reloading a new pattern extends the variable's lifetime and thus may require extra registers.

Some DSP algorithms, such as FIR7, only have one primary input variable, while six temporary variables are used to hold input patterns in the previous six cycles.

These variables are also classified as CPIVS for convenience.

3.1.2 Location of the Dummy Test Operation. If a CO is scheduled at control step *L* (*CS L*), and there are rooms in control steps *L*–1 and *L*+1 simultaneously, we will try to add a test operation in control step *L*–1 first; and if it fails, then we will try to add the operation in control step *L*+1. The reason is that the first pattern of a two-pattern test is an initialization pattern. This pattern produces a spurious operation whose output does not have to be captured into registers, and thus the variable's lifetime is not extended. The location of a test operation pair will affect the test application time. In the proposed method, we will try to reduce overall test application time.

3.2 Overall Flow

In the proposed behavioral synthesis flow, we focus on the datapath part of a circuit. The overall flow is illustrated in Figure 2. It is assumed that the resource constraints (RC) and the scheduled DFG (SDFG) of the circuit are known. We analyze a DFG and find all controllable operations first, and then construct test operation pairs, details of the procedure is explained in Figure 3.

```
Test Synthesis for Delay Fault Testability:
Input: SDFG, Resource Constraints RC
Output: RTL Circuit with transition fault testability

01.  Lifetime Analysis;
02.  Find all controllable operation in SDFG;
03.  Construct TOPs;
04.  if (each module in RC has a TOP) {
05.      Module Allocation;
06.      Register Allocation;
07.      RTL Datapath Generation;
08.      Controller Generation;
09.      Hierarchical Test Generation;
10.  }
11.  else {
12.      add one extra clock cycle and re-schedule;
13.      goto 1;
14.  }
```

Figure 2. Behavioral Test Synthesis Flow

```
TOP Construction procedure:
while (for each modules in RC) {
  // phase I: add one extra test operation
  select the earliest CO scheduled at CS L;
  if (add test operation at CS L–1 success)
    continue;
  else if (add test operation at CS L+1 success)
    continue;
  else {
    // phase II: add two extra test operations
    for (each CS L) {
      if (add TOP at CS L and L+1 success)
        continue;
    }
    // TOP does not exist
    exit;
  }
}
```

Figure 3. TOP Construction

If a test operation pair can be assigned to each functional unit, then exit synthesis flow. Otherwise, we need to schedule the DFG again with one more cycle and repeat the synthesis flow. In most cases, the above procedure is enough to deal with the delay test problem. If a test pair operation still cannot be found in a DFG because of resource constraints, it may be necessary to add a test mode control.

Whenever test operation pairs exist for all functional units, we will perform partial allocation for all test operation pairs, which allocates the operation pair to the same module and assigns the output variable of the second operation to an output register. We apply the Left-Edge Algorithm (LEA) to allocate the remaining operations and variables in the SDFG because this method results in the minimal usage of registers.

Figure 3 illustrates the procedure used to construct test operation pair. In order to reduce overall test application time, the procedure first sorts all controllable operations according to the control step in ascending order. In phase I of the procedure, we select the earliest controllable operation, which is scheduled at control step *L* in the original CDFG, and try to add an extra test operation at the preceding control step (i.e., control step *L*–1). If it does not work out, then a test operation in control step *L*+1 is tried instead. If a test operation cannot be included in either control step, the next possible solution is to add two test operations to test a certain module in phase II. If phase II also fails, there exists at least one module that is not two-pattern testable under the given architecture. We may need to add one extra clock cycle to accommodate the extra TOP, or use a test mode control signal to improve testability.

3.3 An Illustrative Example

An example illustrates the proposed behavioral synthesis flow is presented in this section. Figure 4 gives an SDFG to be synthesized, and e the available resources include an adder and a multiplier. An analysis of the SDFG indicates that there is only one controllable operation *2 and its CPIVS is $\{i_2, i_3\}$. Note that operations +1, +2, and +3 are not controllable since *1 is a multiplication with a constant input, which is not fully controllable. According to the TOP construction procedure, a dummy test operation *2', which is marked by light broken line in Figure 4, is added to CS 3 with $\{i_1, i_4\}$ being its CPIVS. It can be seen that *2 and *2' forms a TOP. The rules for CPIVS selection were discussed in Section 3.2. Note that the dummy test operation added in control step 3 is the second test operation of the TOP for the multiplier module, therefore its must be assigned to the register assigned to the primary output variable (i.e. the variable "out").

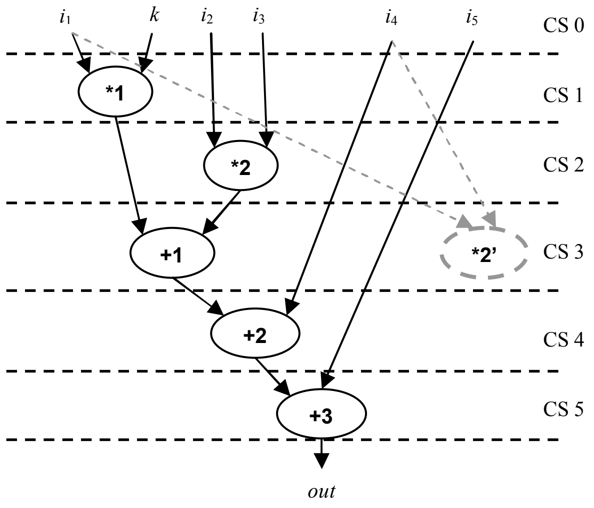

Figure 4. SDFG

It is required to construct an extra TOP for the adder module since there are no controllable addition operations. These two operations are added at control steps CS 1 and CS 2, and their CPIVS are $\{i_1, i_2\}$ and $\{i_3, i_4\}$, respectively. Note that the TOP is not shown in Figure 4. In this example we need the same number of registers compared to Left Edge Algorithm (LEA) based allocation algorithm.

4. Experimental Results

We experimented the proposed method with six circuits, including the example shown in Figure 4 as well as several standard benchmarks such as Diffeq, EWF, FIR7, Wavelet, DCT. For each benchmark, two RTL architectures are generated: one LEA, and the other by the proposed HLTS method. The final circuits are synthesized by Synopsys Design Analyzer with UMC 0.18µm technology.

In this experiment we employ the transition fault model for delay testing. 50000 random test patterns are generated and fed to each circuit, and the patterns are simulated by a transition fault simulator we developed based on HOPE [14]. The rational for 50000 random patterns is that the fault coverage barely changes after applying 50000 random patterns. We also generate deterministic transition fault test patterns for each module (namely, multiplier and adder), and check to see if these patterns can be delivered to the module under test through the circuit hierarchy. The results are shown in Table I for 16-bit datapaths and Table II for 8-bit datapaths.

Four experiments are carried out for each benchmark: LEA(50000) is a circuit generated by LEA and tested by 50000 random patterns, LEA(HIE) is a circuit generated by LEA and tested by hierarchical test patterns. In this case we generate hierarchical test patterns for all operations and choose the results with highest fault coverage. HLTS (50000) is a circuit generated by the proposed HLTS method and tested by 50000 random patterns, and HLTS(HIE) is a circuit generated by the proposed HLTS method and tested by hierarchical test patterns. Column three (RC) gives the resource constraints we used to synthesize the circuits, while column 4 shows the number of control steps in the SDFG for each circuit. Columns 5, 6, 7 give the number of module faults (#MF), detected module faults (#DMF), and module fault coverage (MFC). Module faults are the transition faults appear in the synthesized modules. The last two columns give the silicon area of each circuit and the normalized overhead. The area is the gate count reported by Design Compiler.

The experimental results show that hierarchical test generation for LEA based synthesis method achieves poor fault coverage. The main reason is that the high data dependency and operations with constant input render it difficult to derive hierarchical test patterns from pre-computed test patterns unless TOPs existing in a given CDFG. Furthermore, the proposed method indeed achieves 100% transition fault coverage for all modules in a circuit hierarchy, as it guarantees that a valid test pattern pair can be delivered to the module under test in two consecutive cycles. The proposed HLTS method requires slightly more silicon area. The area penalty ranges from 2% to 11% for 16-bit datapath, and 3% to 23% for 8-bit datapaths.

5. Conclusion and Future Work

We proposed a high-level test synthesis method with integrated hierarchical test generation targeted for delay fault testability. Experimental results show that all transition faults in the functional units are testable. As long as the delay test set for a module is known, the proposed method ensures that this set of patterns can be delivered through the synthesized circuit hierarchy.

References

[1] A. Krstic and K.-T. Cheng, *Delay Fault Testing for VLSI Circuits*, Kluwer Academic Publishers, 1998.

[2] J. Savir and S. Patel, "Scan-based transition test," *IEEE Trans. on CAD*, Vol. 12, No.8, pp. 1232-1241, Aug. 1993.

[3] J. Savir and S. Patel, "Broad-side delay test," *IEEE Trans. on CAD*, Vol. 13, No.8, pp. 1057-1064, Aug. 1994.

[4] M.C. McFarland, A.C. Parker, and R. Composano, "The high-level synthesis of digital systems," in *Proc. IEEE*, pp. 301-318, Feb. 1990.

[5] M. T.-C. Lee, W. H. Wolf, and N. K. Jha. "Behavioral synthesis for easy testability in data path scheduling," in *Proc .ICCAD*, pp. 616-619, Nov. 1992.

[6] M. T.-C. Lee, W. H. Wolf, and N. K. Jha. "Behavioral synthesis for easy testability in data path allocation," in *Proc .ICCD*, pp. 29-32, Oct. 1992.

[7] T. Yang and Z. Peng, "An efficient algorithm to integrate scheduling and allocation high-level test synthesis," in *Proc. DATE*, 1998.

[8] B. T. Murray and J. P. Hayes, "Hierarchical test generation using pre-computed tests for modules," *IEEE Trans. on CAD*, Vol. 9, No. 6, pp. 594-603, Jun. 1990.

[9] S. Bhattacharya and N.K. Jha, "Genesis: a behavioral synthesis system for hierarchical testability", in Proc. *European Design Test Conf.*, pp.272-276, 1994.

[10] I. Ghosh, A. Raghunathan, and N.K. Jha, "Design for hierarchical testability of RTL circuits obtained by behavioral synthesis," *IEEE Trans. on CAD*, Vol. 16, No. 9, pp. 1001-1014, Sep. 1997.

[11] S. Ohtake, H. Wada, T.Masuzawa, and H. Fujiwara, "A non-scan DFT method at register-transfer level to achieve complete Fault efficiency," in *Proc. ASP-DAC*, pp.599-604, 2000

[12] Md. Altaf-Ul-Amin, S. Ohtake, and H. Fujiwara, "Design for Hierarchical Two-Pattern Testability of Data Paths," in *Proc. ATS*, pp.11-16, 2001.

[13] Y. Yoshikaw, S. Ohtake, M. Inoue, and H. Fujiwara, "Design for Testability Based on Single-Port-Change Delay Testing for Data Paths," in *Proc. ATS*, pp. 254-259, 2005

[14] H. K. Lee and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *IEEE Trans. on CAD*, Vol. 15, No. 9, pp. 1048- 1058, Sep. 1996.

Table I. Experimental result for 16-bit datapaths

Circuit	Type	RC	CS	#MF	#DMF	MFC(%)	Area	Overhead
EX	LEA(50000/HIE)	1* 1+	5	4432	3466/39	78.2/0.8	20298.08	1
	HLTS(50000/HIE)	1* 1+	5	4432	4429/4432	99.93/100	22288.56	1.098
FIR7	LEA(50000/HIE)	2* 2+	7	8864	4118 / 0	46.46 / 0	42059.88	1
	HLTS(50000/HIE)	2* 2+	7	8864	8864/8864	100/100	45062.73	1.07
EWF	LEA(50000/HIE)	2* 2+	16	8864	1957/924	22.08/10.42	53189.03	1
	HLTS(50000/HIE)	2* 2+	16	8864	8852/8864	99.86/100	54028.52	1.016
WAVELET	LEA(50000/HIE)	2* 2+	15	8864	5612 / 0	63.31 / 0	64606.80	1
	HLTS(50000/HIE)	2* 2+	15	8864	8864/8864	100/100	71450.16	1.105
DIFFEQ	LEA(50000/HIE)	2* 1+ 1–	5	8864	7821/401	88.23/4.52	40904.09	1
	HLTS(50000/HIE)	2* 1+ 1–	6	8864	8864/8864	100/100	44334.35	1.083
DCT	LEA(50000/HIE)	2* 2+	15	8864	4080/1000	46.03/11.28	54653.36	1
	HLTS(50000/HIE)	2* 2+	15	8864	8848/8864	99.82/100	58338.98	1.067

Table II. Experimental result for 8-bit datapaths

Circuit	Type	RC	CS	#MF	#DMF	MFC(%)	Area	Overhead
EX	LEA(50000/HIE)	1* 1+	5	1048	872/23	83.2/2.19	7244.18	1
	HLTS(50000/HIE)	1* 1+	5	1048	1048/1048	100/100	8345.89	1.15
FIR7	LEA(50000/HIE)	2* 2+	7	2096	1332 / 0	63.07 / 0	15213.65	1
	HLTS(50000/HIE)	2* 2+	7	2096	2096/2096	100/100	17011.11	1.118
EWF	LEA(50000/HIE)	2* 2+	16	2096	824/353	39.31/16.84	21372.04	1
	HLTS(50000/HIE)	2* 2+	16	2096	2096/2096	100/100	21994.75	1.029
WAVELET	LEA(50000/HIE)	2* 2+	15	2096	1761 / 0	84.01 / 0	25282.57	1
	HLTS(50000/HIE)	2* 2+	15	2096	2096/2096	100/100	31041.48	1.227
DIFFEQ	LEA(50000/HIE)	2* 1+ 1–	5	2096	1949/234	92.99/11.16	14697.97	1
	HLTS(50000/HIE)	2* 1+ 1–	6	2096	2096/2096	100/100	16605.35	1.129
DCT	LEA(50000/HIE)	2* 2+	15	2096	1447/456	69.03/21.75	22140.78	1
	HLTS(50000/HIE)	2* 2+	15	2096	2096/2096	100/100	24414.52	1.10