Presented at: *International Test Conference,* Washington, D.C., 1994

HIT – A Hierarchical Environment for Interactive Test Engineering

T. Burch*, J. Hartmann*, G. Hotz*, M. Krallmann*,
U. Nikolaus*, S.M. Reddy†, U. Sparmann*,‡

Technical Report 20/1993
SFB 124–B1

* Fachbereich 14.1 – Informatik, Universität des Saarlandes, D 66041 Saarbrücken, Germany

† Electrical and Computer Eng. Dept., University of Iowa, Iowa City, Iowa 52242, USA

‡ Work partly done while visiting the University of Iowa.

HIT – A Hierarchical Environment for Interactive Test Engineering**

T. Burch*, J. Hartmann*, G. Hotz*, M. Krallmann*,
U. Nikolaus*, S.M. Reddy†, U. Sparmann*,‡

* Fachbereich Informatik, Universität des Saarlandes,
D 66041 Saarbrücken, Germany
† Department of ECE, University of Iowa,
Iowa City, Iowa 52242, USA

Abstract

Conventional tools for test generation and fault simulation appear to the test engineer as black boxes which neither communicate their results in a convenient way, nor allow for any interactive guidance by the test engineer. In contrast, the HIT system presented in this paper supports interactive test engineering, thus combining the power of gate level test generation algorithms with the high level knowledge of the test engineer. Since the HIT system has been integrated into a hierarchical design system (CADIC), the results of test tools can be visualized at the hierarchical circuit specifications given by the designer. Based on this visualization, the critical, untestable areas of the circuit can be easily located. Additionally, the test engineer is supplied with flexible test tools, which allow him to actively guide the test development process. Thus, he can easily apply module specific test strategies or 'communicate' his high level knowledge about the functionality of the overall circuit to speed-up test generation and redundancy identification. An application example shows that with simple strategies for interactive test engineering the results of test generation can be improved dramatically.

1 Introduction

With increasing level of integration the realization of more and more complex algorithms as VLSI circuits is feasible. As a consequence, even today's sophisticated test generation techniques [STS87, SA88, RC90, KP93] may not be able to achieve sufficient fault coverage for such complex combinational designs. As an example, in [Spa92] a fault coverage of only 97.68% was achieved by SOCRATES [STS87] for a 32-bit floating-point multiplier after a running time of over five days on an Apollo DN3000 work station. In addition, it is well known that the last faults in such designs are extremely difficult to detect. Thus, in [Spa92] for the corresponding 16-bit floating-point multiplier by increasing the running time from 4 hours to 35 hours the fault coverage could only be raised from 99.27% to 99.34%, i.e. a gain in fault coverage of only 0.07% had to be paid by an increase in running time of factor 9.

In this case the test engineer has to derive test vectors for faults which were not detected by automated test generation. To achieve this goal, he may have to introduce ad hoc circuit modifications which improve the testability of the design [ABF90, Spa92]. But the information on which faults went undetected, and thus, where testability improvements are most needed is supplied to the test engineer in a very inconvenient manner: as a textual list which is extremely difficult to relate to the hierarchical, graphical descriptions which are usually applied to specify complex designs.

First important steps to solve this dilemma have been reported in [BBH+90, MM93]. Here, the results of automated test generation were made visualizable at a graphical (hierarchical) representation of the circuit. It has been illustrated by examples that based on this visualization the test engineer is enabled to easily locate the parts of the design which are most critical from the testability point of view, and introduce

**This work has been supported by DFG, SFB 124 – "VLSI Entwurfsmethoden und Parallelität" and Grant No. Sp431/1-1.

‡Work partly done while visiting the University of Iowa.

design modifications which improve the controllability and observability of these subcircuits.

The systems described in [BBH+90, MM93] are passive in the sense that they only allow the test engineer to observe the results of test generation algorithms, but not to actually guide these algorithms. In contrast, the HIT (**H**ierarchical **I**nteractive **T**est) system described in this paper is designed to also support interaction between the test tools and the test engineer. Thus, the computation power of test algorithms at the gate level can be combined with the high level knowledge of the test engineer. As a result, complete fault coverage can be achieved for very complex designs even without introducing any design for testability modifications.

A central prerequisite for interactive test development is to give the test engineer an adequate environment, where he can easily convey his ideas without getting distracted by technical details. To achieve this goal the HIT system uses the graphical, hierarchical circuit description given by the designer not only for visualizing results, but also for reading in information from the test engineer which is needed in order to guide the test computation process. Thus, for example subcircuits can be selected graphically for applying module specific strategies, i.e. running test generation or fault simulation only on the selected subcircuit.

In addition to an adequate working environment, we need flexible test tools which allow for guidance by the test engineer. A central idea which will be introduced in this context is that of preassignments for test generation. By such preassignments the test engineer can convey his high level knowledge of how to control or observe embedded modules in order to speed-up automated test generation for them.

Even with guidance by the test engineer there may remain a small subset of faults for which test generation or redundancy identification can not been done automatically, but has to be done manually by the test engineer. The HIT system also gives excellent support in this situation, by for example visualizing for a given input pattern the circuit assignment of values to internal lines, or allowing for graphical identification of redundancies.

In order to illustrate the usefulness of interactive test engineering, we will consider test development for the mantissa part of a floating-point adder as an example. For this circuit purely automated test generation only achieved a fault coverage of 97.77% after a running time of 27 hours on a SUN SPARC ELC work station. With interactive test engineering complete fault coverage could be easily obtained for the circuit without introducing any design for testability modifications. The running time needed for automated test generation and fault simulation in this case was less than one hour.

The paper is organized as follows. Section 2 gives some preliminaries and a detailed motivation for interactive test engineering. The main features of the HIT system supporting interaction between test engineer and test tools are introduced in Section 3. Section 4 shows how to apply these features in order to derive a test with 100% fault coverage for the floating-point adder mantissa part. Finally, the conclusions are given in Section 5.

2 Preliminaries

For a better understanding of the interactive test features supported by the HIT environment it is necessary to shortly introduce the hierarchical graphical design methodology of the CADIC system. This will be done in the first part of this section by using a floating-point adder design as an example[1]. The second part of this section motivates the necessity of interactive test engineering by showing that with purely automated test generation sufficient fault coverage could not be achieved for the example design. How to apply the interactive features of the HIT system in order to obtain a complete test set for the floating-point circuit is discussed in Section 4.

2.1 Hierarchical design example

Most of todays design systems support hierarchical and graphical circuit specification in order to enable the designer to describe large circuits in an elegant and easy to overview manner. The HIT environment for interactive test engineering has been integrated into the hierarchical design system CADIC [HKM86, BHK+87, Mol94]. Thus, all the pictures given in the following will reflect the design level of CADIC. But note that, the methodology for interactive test engineering developed in this paper can be applied equally well to any other hierarchical design system.

Figure 1 shows the top-level description of the mantissa part of a floating-point adder with mantissa width 23 (denoted by $FAM[23]$ in the following) as

[1] For simplicity of presentation we will only consider a slightly simplified version of the mantissa part.

specified in CADIC. All module names have been parameterized with the bit width of their (largest) mantissa input. Since floating point circuits internally compute with a greater precision, the mantissa width is higher for internal modules.

In order to guide automated test generation during interactive test engineering (see Section 4), it is necessary to understand the basic operation of the circuit which will be sketched in the following:
The 'exchange module' ($EX[23]$) exchanges the two operands ($sign_a, mant_a$) and ($sign_b, mant_b$) depending on its left input ex. Input ex is computed by the exponent part such that the operand with the smaller exponent is selected for denormalization. Module $DE[23]$ then executes the denormalization shift, i.e. shifts its input by the absolute value of the exponent difference (de) to the right filling up with zeros from the left. The output width of $DE[23]$ is only $23+3$ since it can be proven that it is sufficient to execute the denormalization shift with an additional accuracy of three bits, where the last bit is computed as the logical OR of all bits which have to be shifted 'over the least significant position' [KM81]. A sign-and-magnitude adder ($SM[26]$) then adds the two mantissas. Its output width is 27 in order to catch a possible overflow. The result is then normalized ($NO[27]$) and the normalization distance (no) is sent to the exponent part for subtraction. Finally, the mantissa is rounded from the internal precision 27 to the external precision 23 by module $RO[27]$. If the mantissa overflows during round-up this must again be signaled to the exponent part (signal mov).

The internal realization of the normalization module $NO[27]$ is given in Figure 2. It consists of a leading zeros counter ($LZ[27]$) which determines the normalization distance no, i.e. the number of leading zeros in the mantissa sum $mant_s$, and a shifter $LS[27]$ which shifts $mant_s$ by no bits to the left filling up with zeros from the right.

In order to minimize the design effort for the submodules of a complex system, they are usually designed with a very regular structure. As an example consider the design of the sign-and-magnitude adder ($SM[26]$). The main part of this adder has been realized following the conditional sum principle [Skl60], where addition is done by the following divide and conquer approach:
Let $a = a_{n-1} \dots a_0$ and $b = b_{n-1} \dots b_0$ be the addition operands (for simplicity we will assume that n is even). Then a conditional sum adder simultaneously computes the sum $s = s_n \dots s_0 = a + b$, and

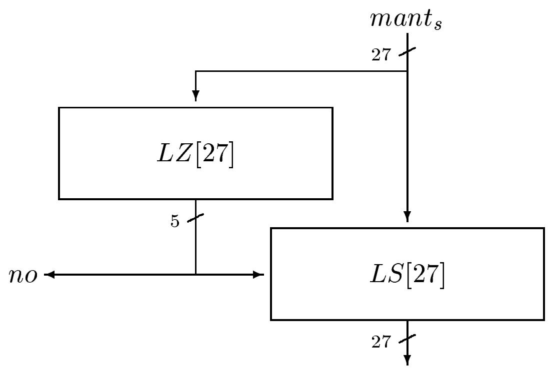

Figure 2: Internal realization of normalization

the incremented sum $s^{+1} = s_n^{+1} \dots s_0^{+1} = a + b + 1$. This is done by applying the recursive scheme shown in Figure 3. Here, $CSA[\frac{n}{2}]$ denotes a conditional sum adder for operand length $\frac{n}{2}$, and $SEL[\frac{n}{2}+1]$ is a circuit consisting of $2 \cdot (\frac{n}{2}+1)$ multiplexers which selects the leading bits of s and s^{+1} depending on the carries resulting from the addition of the least significant bits.

For the specification of a 16-bit adder the designer has to draw the above picture 4 times, namely for 16, 8, 4, and 2 bits. In order to release him from this tiresome task, the CADIC system also supports recursive parameterized circuit descriptions. For our example this means that the designer only has to specify the picture of Figure 3 and the value 16 for parameter n, the system then automatically generates all the necessary pictures for $n \in \{16, 8, 4, 2\}$. For details concerning this *expansion process* refer to [HKM86, BHK+87]. Note, that the above feature of parameterized circuit specification not only eases the design effort, but also allows the designer to develop his own generators for 'macro cells'. As an example, CADIC has been used to implement generators for floating-point adders and multipliers [SFWW89] conforming to the IEEE standard for binary floating-point arithmetic [IEE85].[2]

2.2 Necessity of interactive test engineering

Usually test algorithms appear as black boxes to the test engineer. The only parameter which they accept is an upper bound on the computation time allowed (for example a limit on the number of random patterns for fault simulation, or the number of backtracks per fault for test generation). For small and

[2]The mantissa part of Figure 1 is a simplified version of the 32-bit design generated by the floating-point adder generator.

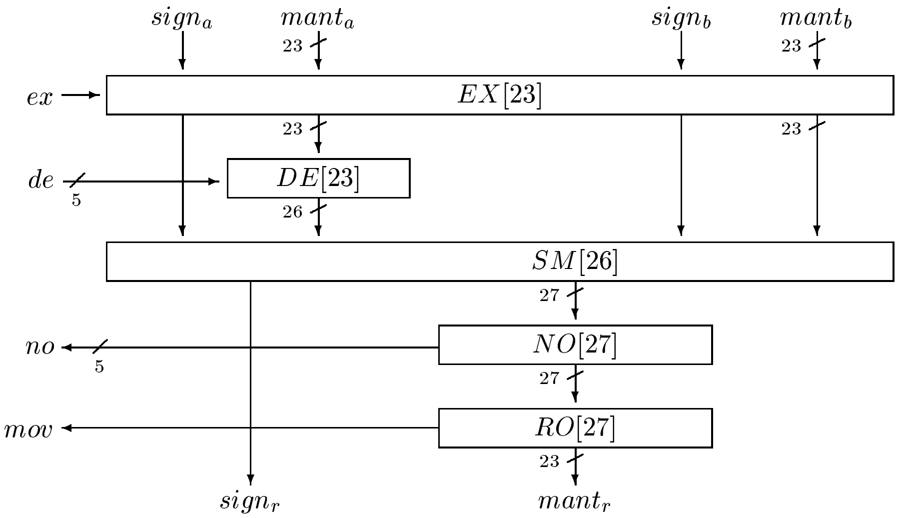

Figure 1: 23-bit mantissa part of floating-point adder ($FAM[23]$)

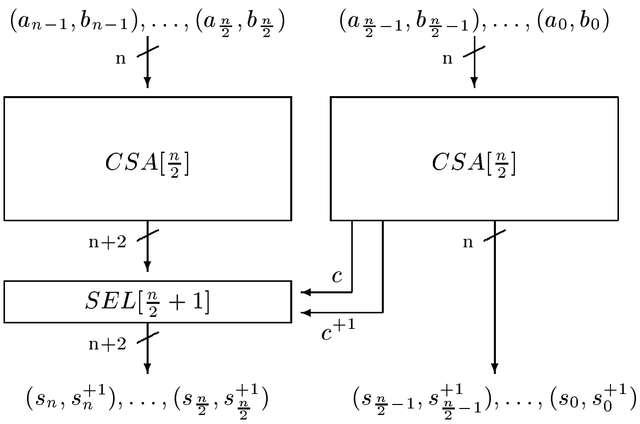

Figure 3: Recursive scheme for conditional sum addition

medium scale circuits like the combinational benchmarks [BF85], or even large circuits with simple structure [BBK89][3] complete fault coverage can be obtained easily by today's sophisticated test generation programs. For large circuits with complex structure there often exists a considerable number of faults for which test generation can not be done in acceptable time. This is not surprising, since the test generation problem is $\mathcal{NP}$-complete [FT82]. A typical example of such circuits where even today's sophisticated test generation algorithms like SOCRATES [STS87] do not deliver sufficient results are floating-point circuit realizations [Spa92].

In such a situation the only available choice for the test engineer is to increase the bound on the time available for test generation. But usually this doesn't help as illustrated by the following example.

Example 1 *Table 1 gives computation times of fault simulation and test generation[4] for the* 23*- and* 52*-bit floating-point mantissa parts* $FAM[23]$ *and* $FAM[52]$ *(see Figure 1).*

exp.	*23-bit*		*52-bit*	
	fc	*CPU-time*	*fc*	*CPU-time*
(1)	*94.93 %*	*6.7 m*	*90.62 %*	*1 h 8 m*
(2)	*97,32 %*	*6.8 m*	*93.45 %*	*1 h 31 m*
(3)	*97.46 %*	*22.4 m*	*93.76 %*	*4 h 34 m*
(4)	*97.66 %*	*2 h 57 m*	*93.91 %*	*36 h 30 m*
(5)	*97.77 %*	*27 h*	*na*	*na*

Table 1: Computation times for test generation

The following experiments have been performed: (1) fault simulation with 320000 random patterns, and (2) ((3),(4),(5)) test generation with 10 (100, 1000, 10000) backtracks. All running times have been measured on a SUN SPARC ELC work station. The fault coverage (fc) is given as the percentage of classified faults, i.e. faults which have been tested or proven redundant. From these figures it becomes clear that the running time needed to classify the last remaining faults increases dramatically. (As an example, when raising the backtrack limit from 1000 to 10000 for $FAM[23]$*, the running time increases nearly by factor 9 while the fault coverage only goes up by* 0.11%*. Thus, for 100000 backtracks, we would expect a running time of approximately 10 days and a fault coverage still less than* 98%*!) As a consequence, even for* $FAM[23]$ *achieving* 100% *fault coverage in acceptable time seems not possible.*

In this situation the only alternative which remains for the test engineer is to apply his high level knowledge about the functionality of the circuit in order to derive a complete test set [Spa92]. But this task is nearly not at all supported by today's automated test generation tools.

The HIT system is aimed at solving this dilemma. It supplies the test engineer with flexible tools, allowing him to combine his high level knowledge about the functionality of the circuit with the computational power of test generation algorithms at the gate level. Thus, 100% fault coverage can be achieved for very complex circuits even without introducing any design for testability modifications. As an example, it will be shown in Section 4 how to apply the features of the HIT system in order to derive a complete test set for the 23-bit mantissa part of Figure 1, based on the high level knowledge given in the previous section.

3 Features for interactive test engineering

The features of the HIT system are aimed at the following two main goals:

1. Give the test engineer an environment which is easy to survey, and allows him to convey his ideas without being distracted by technical details.
2. Supply the test engineer with flexible test tools which allow for interactive guidance in order to speed-up their operation or support him in manual redundancy identification and test derivation.

The first goal is achieved by using the hierarchical graphical description of the circuit not only for visualizing results of test generation, but also to read in information from the test engineer which is necessary in order to guide automated tools. For the second goal we will introduce new methodologies supporting interaction between test tools and the test engineer, as for example: (1) Guiding automatic test generation by preassignments which convey the designers knowledge

[3]For combinational test generation the sequential benchmarks are considered as fully scanned circuits. Thus, additional control and observation points are introduced which extremely simplify test generation.

[4]The fault simulator has been presented in [HSS92], the test generator is based on 16-valued logic [Ake76], and supports high level macros [SMTS89].

of how to control and observe embedded modules. (2) Subcircuit selection for local redundancy identification. (3) Graphical fault list manipulation for marking redundancies which are easy to detect from the 'high level point of view' of the test engineer.

Circuit loading and graphical navigation When starting the HIT system a list of all available circuits is displayed to the user. After selecting a specific circuit all the corresponding hierarchical pictures are loaded into the main memory. For fault simulation and test generation, a flat fault- and netlist is generated. In addition, data structures are computed which allow for efficiently relating objects in the flat representation to their counterparts in the graphical hierarchical representation and vice versa.

After this initialization the topmost hierarchy level is displayed on the screen. It is now possible to navigate through the design with the help of two basic operations: *trace-down* and *trace-up*. When a module is graphically selected with the mouse for trace down, the top level hierarchical representation of its interior is displayed on the screen. Trace-up is the inverse operation and returns to the 'calling picture'. As an example, after loading $FAM[23]$, the picture of Figure 1 is displayed on the screen. If we select trace-down and click the mouse at module $NO[27]$, Figure 2 will be shown. By activating trace-up Figure 1 will be displayed again.

During trace-down it is often difficult to remember the current position in the hierarchical view of the circuit. Thus, there is also a function *global position* which displays the hierarchy of the circuit in a tree structure and highlights the current position. The hierarchy tree for $FAM[23]$ with current module $LS[27]$ is shown in Figure 4.

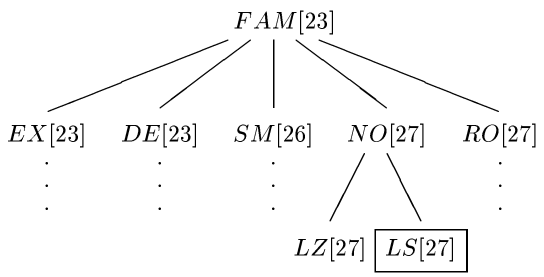

Figure 4: Hierarchy tree

Most of the communication between the user and the system is done over the hierarchical graphical circuit representation. Thus, the above functions trace-down, trace-up, and global position are available for all the operations described in the following paragraphs, supporting tasks like result visualization, assigning values to lines, and subcircuit selection.

Fault list visualization and manipulation If *fault list visualization* is switched on, then the current status of the fault list is displayed in the following form:

(a) For each module the number of unclassified (ucl) and redundant (red) faults inside the module is listed.

(b) For each basic cell (AND, OR, NAND, NOR, NOT) the unclassified and redundant stuck-at faults are noted at the corresponding input and output lines.[5]

As an example, Figure 5 shows the fault status of the normalization module (see Figure 2) after fault simulation and test generation with 10 backtracks.

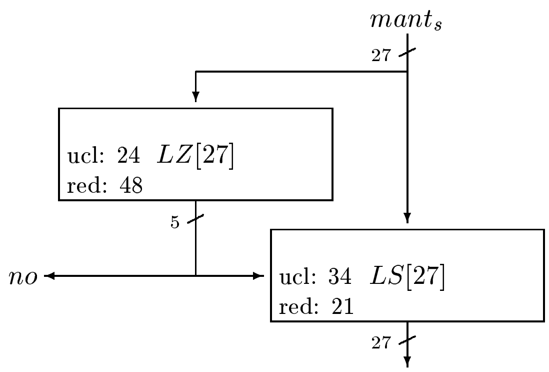

Figure 5: Fault list visualization

The user thus obtains a hierarchical view of the current fault status, which helps him (combined with the trace-down and trace-up operations) to easily identify the critical regions of the circuit with most unclassified or redundant faults. Once switched on, fault list visualization keeps effective until it is switched off again. If the fault status is changed by for example

[5]The test tools of the HIT system not only support the stuck-at fault model for primitive gates (AND, OR, ...), but arbitrary combinational fault models for high level primitives like full-adders, or multiplexers [HSS92]. For simplicity of presentation we restrict to the stuck-at fault model and primitive gates in the following.

test generation or fault simulation, then the results are automatically updated.

In addition the user is also provided with the ability to *graphically manipulate the fault list*. To do this one first selects a basic cell. The status of all faults associated with this cell is then displayed on the screen and can be modified. Thus, the test engineer can classify faults, marking for example an unclassified fault as redundant. As will be seen in the next section, this is quite useful for redundancies which are easy to prove with high-level knowledge about the functionality of the circuit, but difficult to detect from the gate level view of automated test generation.

In regular designs there are often 'regular redundancies', i.e. the same redundancies inside a macro appear at every occurrence of this macro (see Section 4). To relieve the user from the tiring task of classifying these faults in every single occurrence, he can execute the classification for one occurrence, and then have the system perform the same classification for every occurrence of this macro in a graphically selected subcircuit.

Test generation Besides the standard option of determining the number of backtracks, the test generator has been equipped with a feature which allows the test engineer to guide the automated test generator in order to speed-up its computation. This is based on the following observation:

> From his high level point of view the test engineer often knows how to apply desired input patterns to an embedded module, and (or) how to propagate responses. This information may be extremely time consuming to compute at the gate level by automated test generation.

If the user knows that, in order to test an embedded module, some lines (primary inputs as well as arbitrary internal lines) are best set to certain values, he can specify these assignments at the graphical interface. The test generation algorithm will then take advantage of these preassignments. There are two types of *preassignments*:

(a) optional preassignments, and

(b) mandatory preassignments.

Optional preassignments define a starting point for the search process of the test generator. They may be reversed by the algorithm, i.e. a backtrack is executed, if it detects that test construction is not possible with them. Thus, the test engineer can use them to express some vague knowledge. On the contrary, *mandatory assignments* are assumed to be necessary conditions for testing a fault. They are very useful for speeding up redundancy identification but should be applied with care.[6]

Clearly, the concept of preassignments only makes sense if test generation can be restricted to a specific subcircuit (or a single fault). This is achieved by *subcircuit selection* which allows the test engineer to navigate through the circuit and select modules (faults) of interest. Test generation with preassignments is then only executed for the selected faults. Thus, module specific strategies for test derivation can be applied.

Figure 6 shows an example configuration which will be of importance in the next section. The subcircuit consisting of modules $NO[27]$ and $RO[27]$ has been selected for test generation by simply clicking the mouse at these modules. The preassignments (given in square brackets) have been specified by clicking at the corresponding line (bus) and then typing the assignment. In bus assignments x^i is short for $x \ldots x$, i-times, and $*$ denotes that there is no preassignment for the corresponding line. Thus, assignment $[0^{20} * **]$ for the left mantissa means that the leading 20 bits are set to zero and the last three bits are not preassigned.

Preassignments and the corresponding subcircuit selections can be named, stored, and loaded for later reference. In order to enable the test engineer to easily distinguish between optional and mandatory preassignments, they are drawn in different colors.

Fault simulation Standard options for fault simulation are to determine the number of random patterns to be applied to the circuit, or select a dynamic abort criterion (i.e. fault simulation is stopped as soon as a specific number of subsequent patterns does not detect a new fault). In addition, as for test generation, the concept of preassignments combined with subcircuit selection can be used. This enables the test engineer to set some inputs[7] to fixed values which guarantee controllability and observability of an embedded module. The random patterns for fault simulation are then selected such that they respect the preassignments, and

[6]Note, that the concept of preassignments can be easily integrated into any FAN [FS83] based test generation algorithm. As an example, optional preassignments correspond to an appropriate initialization of the backtrack stack.

[7]Preassignments to internal lines are not allowed in this context.

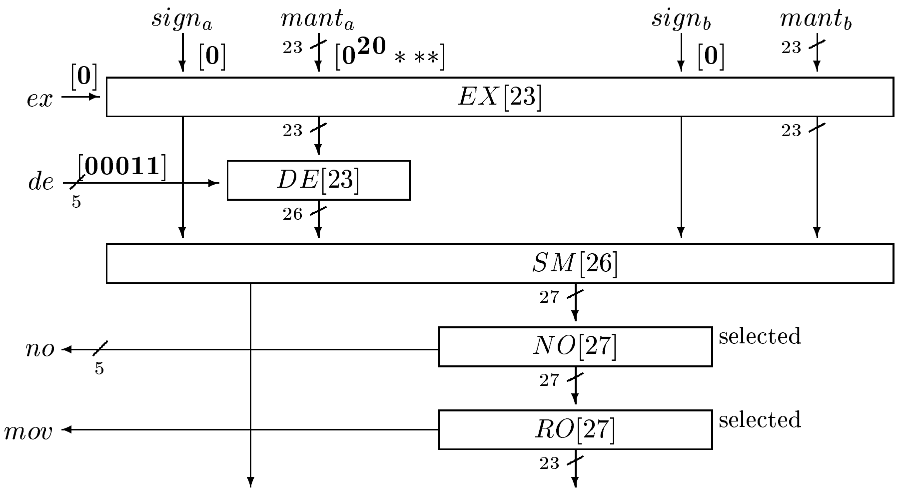

Figure 6: Optional preassignments for test of $NO[27]$ and $RO[27]$

fault simulation is only done for faults of the selected subcircuit.

In order to support the test engineer in the manual construction of test patterns, we have also incorporated the possibility to *visualize overall circuit assignments* which are induced by a specific fault simulation. Here, the test engineer specifies an input pattern to the circuit (this pattern may be partial, i.e. leave some input lines unspecified), and a fault. The assignments which are induced by simulating this fault for the given pattern are then attached to the corresponding lines in the graphical representation. If the test pattern doesn't do the job intended by the test engineer, he can trace through the circuit to find out why fault activation or propagation was not guaranteed, and how the test pattern has to be changed.

Import functions The test development process for a very large design consisting of several submodules, is best done in a hierarchical manner, i.e. we first consider the testability of the isolated submodules and then compose the results to a test for the overall circuit. In order to support hierarchical test derivation we have introduced the *import functions*:

(a) redundancy import, and

(b) test set import.

As will be illustrated in the next section, redundancies which can be easily proven locally, i.e. for an isolated module M, may be much more difficult to identify for automated test generation if M is embedded in a complex system. Thus, a considerable amount of computation time for redundancy identification is saved by first loading the isolated submodules of the overall system, and performing test generation for them. After that the complete circuit is loaded. The local redundancies can then be marked automatically in the overall fault list by simply selecting the *redundancy import* function and clicking the mouse at the corresponding modules.

If the test generation times for the isolated modules are low, we can also group modules together if there are strong dependencies between them which could cause redundancies. A typical situation of this kind is the normalization subcircuit depicted in Figure 2. Here, there are strong correlations between the mantissa input $mant_s$ of the normalization shifter $LS[27]$, and its normalization distance input no determined by $LZ[27]$, i.e. no equals the number of leading zeros in $mant_s$. Thus, it is wise to perform local redundancy identification and import for the combination of both modules (see Section 4).

Assume that the designer knows that controllability and observability of a module are good. Then test generation for the module can be done by first generating a test for the isolated module, and then using this test as a start value for generating a test for the embedded module. The *test set import* function reads in a set of pairs (f, t) where f is a specific fault, and t is a (possibly incompletely specified) test for f. Test generation for each of these faults in the embedded module is then performed with its associated test as an optional preassignment. The test set import function can be combined with preassignments on global lines. Thus, the test engineer may in addition guide the test generation algorithm in how to achieve module controllability and observability.

Design modifications If even with interactive test engineering sufficient fault coverage can not be achieved, ad hoc design for testability modifications have to be introduced. During the test development process the test engineer has learned about the most critical parts of the design, and the reasons for their poor testability. Therefore, circuit modifications improving the testability of the design are often straight forward at this point. Since in our tool design and test development system are totally integrated, it is easy to try different design modifications and their effect on the testability of the overall circuit.

In addition, the careful testability analysis of a design, can also trigger design modifications resulting in small performance improvements. This is due to the fact that redundancies often indicate possible circuit optimizations.

4 Application example

In this section we will describe how the features of the HIT system were used in order to derive a test with 100% fault coverage for a floating-point design. For simplicity of presentation we only consider the circuit FAM[23] as shown in Figure 1.[8] As already indicated in the last section we will proceed in a hierarchical manner, i.e. we first analyze the testability of the isolated main modules of FAM[23], and then use these results for testing FAM[23]. As for Table 1, all running times given in the following have been measured on a SUN SPARC ELC work station.

Analysis of isolated main modules Test generation results for the exchange, denormalization, normalization, and rounding modules are given in Table 2. Note, that we didn't split the normalization module NO[27] into its subcircuits LZ[27] and LS[27] (see Figure 2), since we wanted to detect local redundancies which could be caused by the reconvergent fanout at the inputs of NO[27].

For all modules complete fault coverage could be achieved in less than five minutes. Most local redundancies (red-column) were found in the normalization

module	fc	red	CPU-time
EX[23]	100 %	0	4 s
DE[23]	100 %	8	41 s
NO[27]	100 %	93	4 m 43 s
RO[27]	100 %	3	56 s

Table 2: Test generation results for isolated modules

module (93 redundant faults). Visualizing these redundancies we found that many of them were actually due to the reconvergent fanout at the inputs of NO[27].[9]

The results for the sign-and-magnitude adder are given in Table 3. The first row gives the backtrack limit for test generation, the second and third row specify the resulting fault coverage and the corresponding running time.

# backtr.	10^3 bt	10^4 bt	10^6 bt
fc	99.71 %	99.84 %	99.87 %
CPU-time	3 m	19 m 16 s	17 h 30 m

Table 3: Test generation results for SM[26]

As can be seen from Table 3, the last few faults were extremely difficult to classify by the test generator. Thus, we decided to analyze these faults manually. By *visualizing the results* of test generation we found that the unclassified faults occurred in a very regular pattern, i.e. all of them were located in the multiplexers computing the carry bits inside the SEL modules (see Figure 3). In addition, at lower levels, i.e. near the circuit inputs, the corresponding faults had been proven redundant. This already suggested that the higher level faults were also redundant, but this redundancy was difficult to prove for the test generator. Considering the situation more exactly, we found that the only input patterns which could test these faults locally at the multiplexers had to be of the form shown in Figure 7.

But clearly, if the carry of the sum of two numbers is one, then also the carry of the incremented sum is one. Thus, the input combination of Figure 7 can not be applied, and all the faults could be marked redundant

[8] The procedure given in the following paragraphs can be applied equally well if the mantissa part is not considered as an isolated circuit but with added exponent part.

[9] Note, that these redundancies were not at all obvious from the designers point of view, i.e. not to remove them could not be considered a design fault.

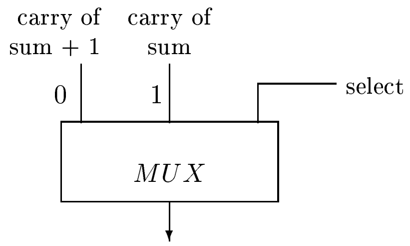

Figure 7: Redundant input for carry computation

with the help of *graphical fault list manipulation*.

Note, that identification of such *regular redundancies* which can be found in any occurrence of a specific subcircuit is often very easy for the test engineer. This is due to the fact that it suffices to prove the redundancy for only one occurrence, and this proof might be straight forward from the high level point of view.

Test computation for the complete circuit Let us now consider test derivation for the complete circuit. As a first step we used the *redundancy import function* to mark all local redundancies in the global fault list. Let us illustrate the advantages of local redundancy identification by an example:

In Table 2 we saw that all local redundancies inside the normalization module $NO[27]$ could be identified in less than 5 minutes. However, when we execute test generation on the complete circuit $FAM[23]$ for these faults, even with 10^5 backtracks and a computation time of approximately 35 hours, 29 of the 93 redundant faults inside $NO[27]$ could not be classified.

After importing the local redundancies we started fault simulation with 320000 random patterns and test generation with 10 backtracks for $FAM[23]$, in order to eliminate all the faults which could be easily detected or proven redundant without any support by the test engineer. Note, that the computation time for this step was less than 14 minutes (see Table 1). The obtained fault coverage at this point was 98.35%, which is already 0.58% more than what the unguided automated test generation achieved for the complete circuit after a running time of 27 hours (see Table 1).

When analyzing the results of this step we found that nearly all of the remaining unclassified faults were located in the modules $NO[27]$ and $RO[27]$. Since these modules are near the primary outputs, fault propagation could not have been a problem for the test generation algorithm. Thus, we conjectured that the poor fault coverage must have been due to the fact that the algorithm had difficulties in generating test inputs for normalization and rounding over the $EX[23]$, $DE[23]$, and $SM[26]$ modules. From the high level point of view this input generation can be seen to be not a problem, and can be done by 'setting one operand to zero and choosing the necessary test input as the second operand'. In order to convey this knowledge to the test generation algorithm, we applied the concept of *optional preassignments*. The preassignments were chosen as shown in Figure 6:
The control input of $EX[23]$ was set such that there is no exchange (= 0), and the signs of both operands were set to + (= 0). The shift distance of the denormalization shifter was set to 3 (= 00011). For the left operand all but the three trailing bits were set to zero ($= 0^{20} * **$).

The effect of these preassignments on the values for the outputs of $EX[23]$, $DE[23]$, and $SM[26]$ is depicted in Figure 8. Here, each a_i (b_i) denotes the value of an input line which has not been preassigned. Note, that with these preassignments only ouputs with a leading zero can be generated at the sign-and-magnitude adder. Thus the preassignments had to be specified as optional, otherwise a fault might have been erroneously classified as redundant.

How dramatically these preassignments simplified the task of the test generation algorithm can be seen from the fact that with these preassignments the fault coverage rose from 98.35% to 99.74% in a computation time of less than 3 minutes. (As a comparison, without the preassignments the test generator only succeeded in raising the fault coverage from 98.35% to 98.6% after a running time of over 18 hours.)

For the remaining faults we first considered the rounding module. Here, we found the situation depicted in Figure 9, i.e. an unclassified stuck-at-1 fault at an AND gate combining the two most significant bits of the normalization module. For testing this fault we need input combination 01 to the gate. But clearly, this input can not be generated by the normalization module, since the output of this module is normalized, i.e. either $0 \ldots 0$ or of the form $1w$, $w \in \{0, 1\}^{26}$. Thus, the fault could be marked redundant.

To find out whether the remaining unclassified faults in the rounding module were due to the same effect, we proceeded as follows: (1) An input vector was derived manually which applied the vector $0 \ldots 0$ to the rounding module. We then executed fault simulation, thus testing all faults which are testable by the all zero input. (2) We then started test generation where the optional preassignments were supplemented by a *mandatory preassignment* setting the leading output

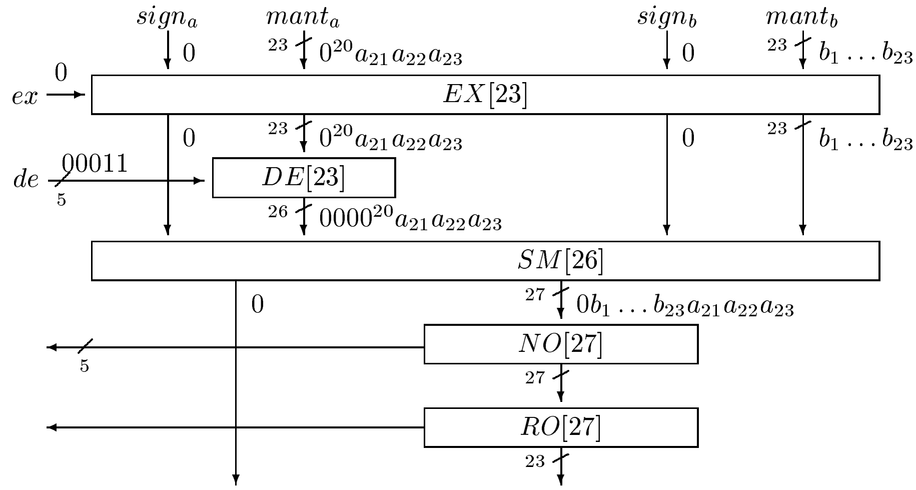

Figure 8: Effect of preassignments

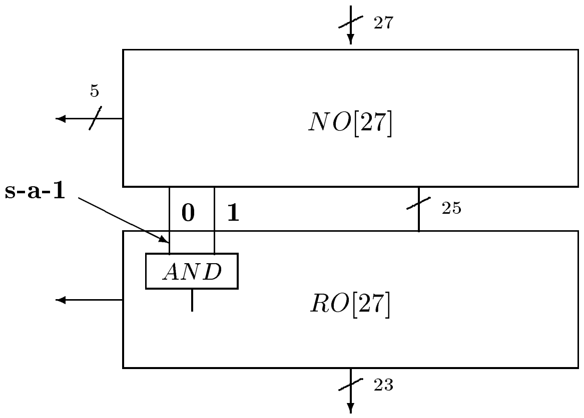

Figure 9: Redundancy inside rounding module

bit of the normalization module to 1. (The correctness of this procedure follows directly from the fact that any input $\neq 0 \ldots 0$ to $RO[27]$ must start with a leading one.) As a result, all the remaining unclassified faults inside the rounding module were classified in a computation time of less than a minute.

Finally, there were only 7 unclassified faults left. For 6 of them tests could be generated with the help of preassignments. The last fault was redundant, since it could not be propagated due to rounding. We omit the details for brevity.

Thus, the same test tools which only achieved a fault coverage of 97.77% after a computation time of 27 hours (see Table 1), could be used interactively to obtain a complete test set. The time needed for automated test generation and fault simulation in this case was less than 1 hour[10].

5 Conclusion

In this paper we have presented the HIT system for interactive test engineering. The main objective of this system is to give the test engineer the possibility to combine his high level knowledge about the functionality of the circuit with the power of gate level automated test generation techniques in order to improve test generation results in cases where automated techniques alone do not deliver sufficient fault coverage. As a 'by-product', the HIT system also helps the test engineer to find out the best places for design for testability modifications in case that even with interactive test engineering satisfactory results can not be obtained.

Since the HIT system is integrated in a graphical hierarchical design environment (the CADIC system), all communication between the test engineer and the system (visualization of results, selection of subcircuits, assignment of values to lines) can be done over the graphical representation of the design. Thus, the test engineer is not distracted by technical problems like associating a textual fault list with the graphical

[10]We exclude the 10^6 backtrack measurement for $SM[26]$ since it was not necessary for the test development process and only presented here to clarify the difficulties of automated test generation in proving the corresponding faults redundant.

description of a design. In addition we have shown that there are simple ways to increase the flexibility of test tools. With such flexible tools the test engineer is enabled to guide automated test generation based on his high level knowledge about the circuit. Two very important concepts in this context are preassignments for test generation, and local redundancy identification and import. Finally, the HIT system also gives excellent support for manual test construction and identification of regular redundancies by visualizing the assignments of internal lines for fault simulation and allowing for graphical fault list manipulation. The usefulness of interactive test engineering has been demonstrated by deriving a test with 100% fault coverage for the mantissa part of a floating-point adder where automated test generation alone was not able to achieve sufficient fault coverage.

The graphical features of the HIT system can not only be applied to support interactive test engineering, they are also extremely useful for the development and improvement of test tools. Thus, the hierarchical graphical circuit representation can also be applied for debugging algorithms or visualizing the effect of newly developed heuristics. These applications are currently under investigation.

Acknowledgements

The authors would like to thank P. Molitor and B. Schieffer from the Universität des Saarlandes for their continuous support and helpful discussions. In addition, we want to acknowledge B. Becker, R. Hahn, R. Krieger, and R. Drechsler from the J.W. Goethe-Universität, Frankfurt for helpful discussions and supplying us with the testcirc environment.

References

[ABF90] M. Abramovici, M.A. Breuer, and A.D. Friedman. *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

[Ake76] S.B. Akers. A logic system for fault test generation. *IEEE Transactions on Computers*, C-25:620–630, June 1976.

[BBH+90] B. Becker, Th. Burch, G. Hotz, D. Kiel, R. Kolla, P. Molitor, H. G. Osthof, G. Pitsch, and U. Sparmann. A graphical system for hierarchical specifications and checkups of VLSI circuits. In *Proceedings of the 1st European Design Automation Conference*, pages 174–179, 1990.

[BBK89] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles for sequential benchmark circuits. In *Proceedings of the International Symposium on Circuits and Systems*, pages 1929–1934, 1989.

[BF85] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *Proceedings IEEE International Symposium on Circuits and Systems*, 1985.

[BHK+87] B. Becker, G. Hotz, R. Kolla, P. Molitor, and H.G. Osthof. Hierarchical design based on a calculus of nets. In *Proceedings of the 24th Design Automation Conference*, pages 649–653, June 1987.

[FS83] H. Fujiwara and T. Shimono. On the Acceleration of Test Generation Algorithms. *IEEE Transactions on Computers*, C-32(12), December 1983.

[FT82] H. Fujiwara and S. Toida. The complexity of fault detection problems for combinational logic circuits. *IEEE Transactions on Computers*, C-31:555–560, June 1982.

[HKM86] G. Hotz, R. Kolla, and P. Molitor. On network algebras and recursive functions. In *Proceedings of the 3rd International Workshop on Graph Grammars and Their Applications to Computer Science, LNCS 291*, pages 250–261. Springer Verlag, 1986.

[HSS92] J. Hartmann, B. Schieffer, and U. Sparmann. Cell oriented fault simulation. In *Proceedings of the European Simulation Multiconference 92*, pages 424–429, 1992.

[IEE85] The Institute of Electrical and Electronics Engineers, Inc. *IEEE Standard for Binary Floating-Point Arithmetic ANSI/IEEE Std 754-1985*, 1985.

[KM81] U.W. Kulisch and W.L. Miranker. *Computer Arithmetic in Theory and Practice.* Academic Press, 1981.

[KP93] W. Kunz and D.K. Pradhan. Accelerated dynamic learning for test pattern generation in combinational circuits. *IEEE Transactions on CAD*, pages 684–694, May 1993.

[MM93] J. Moorman and S.D. Millman. Visualizing test information: A novel approach for improving testability. In *Proceedings of the 1993 International Test Conference*, 1993.

[Mol94] P. Molitor. A hierarchy preserving hierarchical bottom-up 2-layer wiring algorithm with respect to via minimization. *Integration, the VLSI Journal*, January 1994.

[RC90] J. Rajski and H. Cox. A method to calculate necessary assignments in algorithmic test pattern generation. In *Proceedings of the 1990 International Test Conference*, pages 25–34, 1990.

[SA88] M.H. Schulz and E. Auth. Advanced automatic test pattern generation and redundancy identification techniques. In *Proceedings of the* 18*th Symposium on Fault-Tolerant Computing*, 1988.

[SFWW89] U. Sparmann, T. Fettig, G. Wannemacher, and W. Weber. Entwurf von kombinatorischen Schaltkreisen für schnelle Gleitkommaarithmetik. 50 Seiten, Documentation of CADIC Designs, Universität des Saarlandes, 1989.

[Skl60] J. Sklansky. Conditional-sum addition logic. *IRE-EC*, 9:226–231, 1960.

[SMTS89] T.M. Sarfert, R. Markgraf, E. Trischler, and M.H. Schulz. Hierarchical test pattern generation based on high-level primitives. In *Proceedings of 1989 International Test Conference*, pages 470–479, 1989.

[Spa92] U. Sparmann. Derivation of high quality tests for large heterogeneous circuits: Floating-point operations. In J. Buchmann, H. Ganzinger, and W.J. Paul, editors, *Teubner Texte zur Informatik (Band 1): Festschrift zum* 60*. Geburtstag von Günter Hotz*, pages 425–439. Teubner Verlag, 1992.

[STS87] M.H. Schulz, E. Trischler, and T.M. Sarfert. SOCRATES: A highly efficient automatic test pattern generation system. In *Proceedings of the International Test Conference*, pages 1016–1026, September 1987.