System Diagnostics in HW-SW Codesign [1]

Gy. Csertán
Department of Measurement and Instrument Engineering
Technical University of Budapest, Hungary
E-mail: csertan@mmt.bme.hu

Abstract

In this paper a novel approach is presented, which can successfully be used for the underlying hierarchical modeling of HW-SW codesign during the whole design cycle. This new method combines the conventional, performability evaluation oriented description of the functional units of the system with the description of fault effects and error propagation.

Various dependability measures can be extracted from the extended system model. This work deals with diagnostic design, that is the iterative process of model construction, test generation, concurrent fault simulation and integrated diagnostics. The result is an optimized, ordered test set of the system.

1 Introduction

One of the most promising design automation methods is HW-SW codesign (Figure 1 unshaded area), that denotes "the joint specification, design, and synthesis of mixed HW-SW systems" [3].

Unfortunately existing HW-SW codesign implementations (like Ptolemy, COSMOS, SpecSyn [7]) lack of integrated support for diagnostic design (test generation, testability analysis). This becomes crucial in safety related applications, like process control and automation. In [8] a method is presented for testability analysis as part of integrated diagnostics, but the problems of generating the input model of this method, designing of the test set and dealing with diagnostic uncertainty remain still unsolved.

The aim of our work is the development of a tool-box for model-based diagnostics and dependability evaluation in the form of an extension of the existing functional design tool Ptolemy. The basic models and technologies developed are fully coherent with those used in the original tools in order to keep the integrity of the design environment and to avoid unnecessary model transformations.

In this paper a novel approach is presented, that uses the dataflow notation as the modeling methodology of HW-SW codesign. Using this approach the behavior of the functional units of digital computing systems can be hierarchically described and aspects of faults, their effects, and error propagation can be handled during the design process. As it is shown in [4] the following problems can be solved concurrently with system design:

- fault simulation
- test generation
- estimation of optimal diagnostic strategies
- testability analysis for both built-in and maintenance tests
- failure modes and effects analysis (FMEA), risk analysis

[1] This research is supported from: EC Research Project #CP94:9624 FUTEG; Hungarian-German Joint Scientific Research Project #70; German Scientific Research Project SFB-182

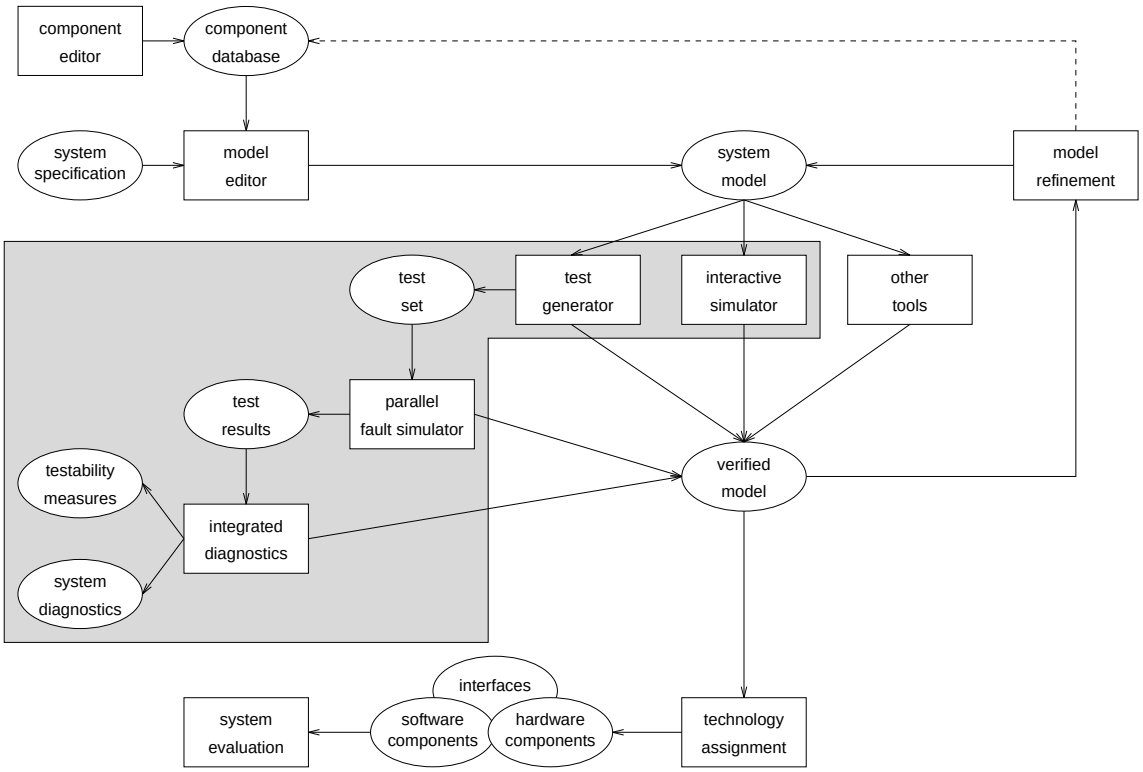

Figure 1: HW-SW codesign process

The work consists of four parts. After the introduction in Section 2 the modeling approach is presented: first a general description, then the selected dataflow notation is introduced, finally the theoretical background of model refinement is described. In Section 3 a simple example is given. Finally Section 4 presents an overview of the implemented design environment and gives a few concluding remarks.

2 The Modeling Approach

Since the application area of HW-SW codesign is very broad, no general solution can be given. We restrict our activities to the field of digital computing and control systems, where dependability is of primary concern. Within this field we will focus on the first step of HW-SW codesign (shaded area in Figure 1). This is the iterative process of model construction, model evaluation and model refinement, that is executed until the parameters of the model meets the requirements.

In the first step of HW-SW codesign the system is modeled by extended dataflow networks. At this level of abstraction only the flow of data is modeled in the form of token flows without any description of the data transformation performed by the components (uninterpreted modeling). Primarily performance analysis and formal analysis are aimed at.

Through stepwise refinement more and more structural and functional details are incorporated into the initial model. As the system's structure and the data processing functions of the components are increasingly exactly defined the inherent uncertainty of the model decreases, leading to more exact analysis results. Test generation and testability analysis can be done in this phase. Test generation at this level delivers the system-level diagnostic tests set of the system.

Finally when all functional details become fully specified (interpreted modeling) the second

step of HW-SW codesign: system partitioning and automated hardware and software synthesis can be started. After technology assignment a backannotation step has to be done. It feeds back information from the implemented system to the functional model, and assigns the functional components to HW and/or SW components. As a result, common mode faults have to be taken into account in the model (two or more functional units are implemented by the same HW or SW component) and a refinement step can be done, that further restricts the nondeterminism within the model (by removing firing rules within the nodes). Test generation and testability analysis is still possible, and the result of test generation is the low level test set of the components, e.g. self test of a component.

Note, that this way even the modeling of the fully designed system is possible, but there are more effective approaches for the evaluation of this low level model (see later).

Our approach is based on the idea of modeling the fault effects and their propagation similarly to the flow of data in the functional model. Tokens representing the data can be colored either as correct or as faulty. A set of error propagation paths can be estimated by tracing their flow from the fault site to the outputs of the network. Diagnostic uncertainty is introduced in order to express conditional error propagation. This way the simulation and test generation algorithm delivers a superset of propagated fault effects in the system, but in the model all potential consequences of a fault are incorporated. In subsequent steps of model refinement this global overview of the system effectively supports test generation procedures by radically restricting the search space to the solutions of those from the coarse model [5].

Of course the whole process of model refinement is only useful if the results from a previous design phase can really be used to guide the evaluation of the current phase model, leading to the mentioned restricted search space. Therefore the rules of model refinement has to be defined.

A more detailed fault model and a more precise description of the reactions of functional units to erroneous input values can be defined by using multi-valued logic. For example, the tokens and component fault states can be qualitatively grouped (colored) according to the severity of the user-defined fault effects like:

- catastrophic (causes a damage in a component)
- fatal (blocking the further operation, e.g. an undetected wrong opcode input of a CPU-like element)
- incorrect (may invoke only error propagation, like wrong input data to be processed by the CPU)

It should be pointed out, that the use of other guiding attributes in this user-defined colorings of the tokens and propagation rules offers full freedom for the analysis of different user requirements.

2.1 Dataflow Networks

Asynchronous dataflow networks, introduced in [6] combine a graphical representation (dataflow graphs) and a formal mathematical background. The former leads to an easily edit-able and understandable model, while the latter makes formal analysis of the model possible. In comparison with Petri nets, dataflow models are more compact, they correspond to the well known black-box approach, the way the designer thinks during system design. On the other hand the evaluation of dataflow models is not limited by complexity problems as in case of process algebraic approaches.

Therefore the dataflow notation is well-suitable for conceptual modeling of computing systems in the early design phases [2], for early validation of computing systems, for performance evaluation, and as modeling language in HW-SW codesign [4].

A *dataflow network* N is a set of nodes P_N, which execute concurrently and exchange data (tokens) over point-to-point communication channels C_N. *Dataflow nodes* represent the functional elements of the system and describe their signal propagation behavior by relations between input and output, eventually depending on the previous state of the node. The use of relations instead of input-output functions allows the modeling of non-deterministic behavior. The *channels* of the dataflow network symbolize the interaction between the functional elements of the system. Each channel connects one output of a node to one input of a node.

Definition 1 *A* token set M *is a set of message types, including also the empty set, which means that no message is sent, and the token* x*, which represents uncertainty.*

Definition 2 *A* dataflow network *is a tuple* $DFN = (N, C, S, s_0)$ *where:*

C *- set of channels (I-input, O-output, and IN-internal channels)*
N *- set of nodes*
S *- set of states, composed from the states of nodes and channels*
s_0 *- initial state,* $s_0 \in S$

A channel may contain any sequence of tokens, the state of a channel is $S = M^*,\ s_0 = \emptyset$.

Definition 3 *A* dataflow node n *is a tuple* $(I_n, O_n, S_n, s_n^0, R_n)$ *where:*

I_n *- set of input channels*
O_n *- set of output channels*
S_n *- set of states*
s_n^0 *- initial state,* $s_0 \in S$
R_n *- set of firings,* $r_n \in R$ *is a tuple* (s, X_{in}, s', X_{out})
s, s' *- states before and after the firing,* $s, s' \in S_n$
X_{in} *- input sequence,* $X_{in} : I_n \mapsto M$
X_{out} *- output sequence,* $X_{out} : O_n \mapsto M$

The functional behavior of a node n is defined by the set of firing rules R_n over the input domain and over S_n, the set of possible states of the node. A node is ready to execute as soon as the data required by one of its firing rules are available and the node is in a proper state. The meaning of firing rule $r \in R_n$, denoted by $r = (s, X_{in}, s', X_{out})$ is that if the node n is in state $s \in S_n$, each of the input channels $i \in I_n$ holds at least $X_{in}(i)$ data items, then firing rule r is potentially selected for execution. The execution of firing rule r removes $X_{in}(i)$ data items from each input channel $i \in I_n$ and outputs $X_{out}(j)$ data items on each output channel $j \in O_n$, while the node changes its state from s to s'.

2.2 Extension of the Model

Similarly to Petri nets, there are many ways to extend the dataflow notation: Introduction of the notion of time in order to model the duration of different activities of the node. The definition of

firing rules allows to express nondeterministic behavior of the nodes. In many cases it can lead to undesirable effects. To prevent such cases priority can be added to the firing rules. On the contrary, when nondeterministic behavior is requested, occurance probability can be added to the firing rules.

Our extension is different: the set of tokens (data items), set of node states, and the set of firing rules is extended. Thus it is possible to describe the behavior of the system even in the faulty case: modeling of fault occurance, fault propagation, fault removal, fault detection, etc. This extension leaves the structure of the model unaltered. Of course the previously mentioned extensions can be combined with this behavioral extension.

2.3 Refinement of the Model

As mentioned in the previous section, during model refinement the designer has to meet some rules in order to make sure, that the results gained by the evaluation of the first phase model are consistent with the results gained for models from later design phases. It will be shown, that if the second model (DFN) is a refinement of the first model (DFN), the required consistence can be assured.

Note: In the definitions and proves we will use the assumption, that on each channel only one token is sent or consumed at the same time. It does not restrict the usability of the presented method, since it can easily be generalized for the case, when more then one token is moved. In the paper relations over sets are rather presented as functions over the power sets of the set. For example relation R from X to Y is denoted by: $R : X \mapsto Y^*$, where Y^* is the power set of Y.

Definition 4 *A* nondeterministic finite-state transducer *(NDFST) is a tuple* $NDFST = (\Sigma, , , S, S_0, \delta, \omega)$ *where:*

Σ - *input alphabet*
$,$ - *output alphabet*
S - *set of states*
S_0 - *initial states,* $S_0 \subseteq S$
δ - *state transition function,* $\delta : S \times \Sigma \mapsto S^*$
ω - *output function,* $\omega : S \times \Sigma \mapsto , ^*$

A dataflow node can be described by a NDFST using the following substitution:

$\Sigma = M^i$, $i = |I_n|$
$, = M^o$, $o = |O_n|$
$S_0 = s_n^0$
$\delta = \{\ldots, (r_i(s), r_i(X_{in}), r_i(s')), \ldots\}, \forall r_i \in R_n$
$\omega = \{\ldots, (r_i(s), r_i(X_{in}), r_i(X_{out})), \ldots\}, \forall r_i \in R_n$

Now the computation of a dataflow network can be described by composition of NDFSTs, which in turn is a composition of relations:

Definition 5 *The* computation of a dataflow network *is a relation* $CP : IM \times S \mapsto OM^*$, *where*

IM - *input mapping,* $IM = (M^*)^i$, $i = |I|$
OM - *output mapping,* $OM = (M^*)^o$, $o = |O|$

Definition 6 *The* set of faults F *contains all possible fault hypotheses under the given fault model.*

F defines the set of faults, for which tests have to be generated, and in presence of which the testability measures of the system have to be evaluated. For example in case of a single-fault model, the elements of F are all possible single-faults of the system. If each component has only one fault, the number of hypotheses in F is equal to the number of components.

Definition 7 *A* test *for fault $i \in F$ is an ordered pair $t_i = ((v, is), tr)$, where:*

$v \in IM$ *- test vector*
$is \in S$ *- initial state (with fault i)*
$tr \in OM^*$ *- test result*

The test set of the system is denoted by $T = \{\ldots, t_i, \ldots\}$, and clearly $T \subset CP$. $cp_x = ((v, is), tr)$ is a test iff om, delivered by the fault free computation $cp_y = ((v, s_{ff}), om)$, and tr differ. s_{ff} denotes the fault free initial state of the system that is gained from is by removing the assumed faults of the fault hypothesis. om and tr differ iff $\forall i \in om$ and $\forall j \in tr$ $i \neq j$.

Since each test is a relation, the difference of test results, and the response of the fault free system can only be tested by pairwise comparison. It means, that no matter which outcome a test and the fault free system will deliver, they should be different.

Definition 8 *Test t_i is called a* certain/uncertain *test for fault i, if $\forall i \in om$ and $\forall j \in tr$ differ not only/only by x tokens. A fault is* non detectable/uncertainly detectable/certainly detectable *if no test exists/an uncertain test exists/a certain test exists for it. The set of non detectable faults/uncertainly detectable fault/certainly detectable fault is denoted by F_n, F_u, F_c. $F_c \subseteq F_u$, $F_u \cap F_n = \emptyset$, and $F_u \cup F_n = F$.*

Since uncertainty is present in the modeling, the exact value of testability measures can not be elaborated. Instead a minimum and a maximum value can be computed, and the exact value will be somewhere in between. In this case the set of certainly detectable faults represents the minimum value (this is called the pessimistic case) and the set of uncertainly detectable faults represents the maximum value (it is called the optimistic case).

Changes in the model (switching from one level of abstraction to another one) are described by refinement, which can be any be any of:

1. modification of the structure:
   - changing the connections among nodes
   - adding new channels to the network
2. modification of the behavior of the components:
   - changing the token set and the firing rules
   - altering the set of states and the firing rules

The refinement of the network defines the rules a designer has to meet during the design in order to have "monotonously changing testability measures". In this work we only deal with the refinement of the behavior of the network (point 2 in the above list).

Definition 9 *DFN^2 is a* refinement *of DFN^1, if the structure of the network is unchanged, M^2 is a refinement of M^1, $\forall n \in N$ S_n^2 is a refinement of S_n^1, and $R_n^2 \subseteq \mathcal{R}(R_n^1)$. Since the set of fault hypotheses is closely related to the set of states, F^2 is a refinement of F^1.*

$\mathcal{R}$ describes the refinement of firing rules: Let suppose a firing rule r^1 and its refinement r^2. $r^1 : (s_{pre}^1, im^1) \mapsto (s_{post}^1, om^1)$, where $r^1 \in R_n^1$, $s_{pre}^1, s_{post}^1 \in S_n^1$, $im^1 \in (M^1)^i$, $om^1 \in (M^1)^o$. The refinement of tokens transforms M^1 to M^2 such that $m^1 \in M^1 \mapsto \{\ldots, m_i^2, m_j^2, \ldots\} \subseteq M^2$. The refinement of states transforms S^1 to S^2, such that $s^1 \in S^1 \mapsto \{\ldots, s_i^2, s_j^2, \ldots\} \subseteq S^2$. Now r^2, one possible transformed of r^1, is a mapping $r^2 : (s_{pre}^2, im^2) \mapsto (s_{post}^2, om^2)$, such that if $s_{pre}^1 = s^1$ then $s_{pre}^2 \in \{\ldots, s_i^2, s_j^2, \ldots\}$ and if $t^1 \in m^1$ is a token consumed/produced by r^1, the token $t^2 \in \{\ldots, m_i^2, m_j^2, \ldots\}$ has to be consumed/produced by r^2. These rules do not effect uncertain states and tokens: if s_{pre}^1/s_{post}^1 is uncertain, then s_{pre}^2/s_{post}^2 can be any $s \in S^2$, and if token t^1 is uncertain, then t^2 can be any $t \in M^2$. The set of possible transformed firings $Ref(r_1)$ is made by combining of any possible $s_{pre}^2, s_{post}^2, im^2, om^2$ and the set of transformed firings will be a subset of it $r^2 \subseteq \mathcal{R}(r^1)$.

Theorem 1 *$L1$ and $L2$ with the corresponding dataflow networks DFN^1 and DFN^2 are two different levels of modeling. If DFN^2 is a refinement of DFN^1, then*

$$\frac{|F_c^2|}{|F^2|} \geq \frac{|F_c^1|}{|F^1|} \text{ and } \frac{|F_u^2|}{|F^2|} \leq \frac{|F_u^1|}{|F^1|}$$

Proof The theorem is proven in three steps: 1. it will be shown, that if f^1 is certainly detectable, then $\forall f^2 \mid f^2 \in \{\ldots, f_i^2, f_j^2, \ldots\}$ is also certainly detectable. 2. if f^1 is non detectable, then $\forall f^2$ is also non detectable. 3. if f^1 is uncertainly detectable, then $\forall f^2$ is either non detectable, or certainly detectable, or uncertainly detectable. If these three steps are proved, the statement of the theorem is proved.

Step 1: According to Definition 9, if a firing rule at $L1$ delivers a non x token, the corresponding firing rules at $L2$ will also deliver a non x token. Exploiting the composition of nodes, it can be proven by induction, that if DFN^1 produces some non x tokens (t^1 is a certain test by Definition 8), than DFN^2 also produces non x tokens, thus $\forall t^2$ corresponding to t^1 is certain test and f^2 is certainly detectable.

Step2: The detailed prove is omitted since it can be done similarly than in Step 1, except that here we have to exploit the the similarity of computations without and with the fault hypothesis (Definition 7) and we have to argue about the absence of x tokens in the response of the DFN.

Step3: If f^1 is uncertainly detectable, then the test result and the result of fault free computation differ only by x tokens (see Definition 8. If firing r^1 delivers x tokens, then according to Definition 9 the firings r^2 can deliver both x and non x tokens. Again using the composition of nodes it can be shown by induction, that when x tokens in r^1 are refined to non x tokens in r^2, f^2 gets either certainly detectable (test and fault free computation results differs by not only x tokens), or non detectable (test and fault free computation results do not differ anymore) When x tokens in r^1 remain x tokens in r^2, f^2 remains uncertainly detectable. □

Informally, the theorem states, that after refinement of the model, the relative number of detectable faults under pessimistic assumptions will be larger, while the relative number of detectable faults

under optimistic assumptions will be smaller. It means, that by adding more and more information to the model (removing uncertainty), the testability measures of the system can be evaluated more and more precisely.

3 An Example

Because of space limitation the example is kept very simple. It is a single node, with one input and one output. It represents a reference signal generator. The input comes from a power supply, and the output delivers the reference signal. First the normal behavior is described, then the extended behavior is given under a very simple fault assumption. The graphical representation of the node is given in Figure 2.

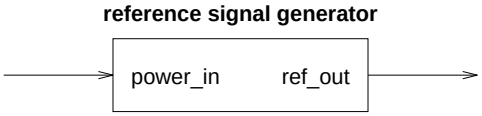

Figure 2: DF graph of the reference signal generator

3.1 Basic Component (fault-free)

In the fault-free case the dataflow node receives a single token on its input and delivers a single token to its output. It has only one state, `s0`. The behavior is described by the firing:

```
r0={s0,power_in,s0,ref_out}
```

The meaning of the firing is, that from `s0`, if a token is received on the input channel `power_in`, the node goes into `s0` and produces a token on the output channel `ref_out`.

3.2 Extended Component (faulty)

In the faulty the tokens are colored as follows:

ok – denotes fault-free information
fty – denotes faulty information
unc – denotes uncertainty (information is either faulty or fault-free)

The only fault related functional extension of the component is fault propagation in fault-free (state `s0`) and in faulty (state `s1`) states. Thus the set of firings is the following:

```
r0={s0,power_in=ok,s0,ref_out=ok}     r2={s1,power_in=ok,s1,ref_out=fty}
r1={s0,power_in=fty,s0,ref_out=unc}   r3={s1,power_in=fty,s1,ref_out=fty}
```

The meaning of the firings is: `r0` denotes the original function. `r1` describes the behavior of the fault-free component if faulty supply voltage is received from the power supply; the reference signal might be correct or incorrect. `r2` describes the behavior of the faulty component in case of fault-free inputs; the reference signal is incorrect. Finally `r3` describes the behavior of the faulty component in the case if the input voltage is faulty; the reference signal is incorrect.

Remember that the structure of the model remains unchanged, but the cardinality of the token set increases from 1 to 3, the size of the state space of the node increases from 1 to 2, and the number of firing rules increases from 1 to 4.

3.3 Refined Component

The refinement rules state, that during refinement the structure of the model must not be altered and for each node of the network the sets (tokens, states, and firing rules) after the refinement step have to be a refinement of the corresponding sets before the refinement step. Of course replacing a node by a subnetwork, does not count as a structural modification, thus it is an allowed operation during the refinement. The impact of these rules is shown on the simple example. In the refinement step the set of tokens is changed:

ok – correct signal value
low – signal value is lower than correct
high – signal value is higher than correct

The set of faulty tokens is now split, and the token for expressing uncertainty is no more needed (uncertainty vanished from the model). The states of the nodes changed similarly: `s0` denotes the fault-free state, in state `s1a` the component delivers low reference signal, and in state `s1b` it delivers high reference signal. Therefore the new firing rules are the following:

```
r0'={s0,power_in=ok,s0,ref_out=ok}        r5'={s1a,power_in=high,s1a,ref_out=low}
r1'={s0,power_in=low,s0,ref_out=low}      r6'={s1b,power_in=ok,s1b,ref_out=high}
r2'={s0,power_in=high,s0,ref_out=ok}      r7'={s1b,power_in=low,s1b,ref_out=low}
r3'={s1a,power_in=ok,s1a,ref_out=low}     r8'={s1b,power_in=high,s1b,ref_out=high}
r4'={s1a,power_in=low,s1a,ref_out=low}
```

After refinement, firing `r0` remained unchanged and it is denoted by `r0'`. Firing `r1` is split into `r1'` and `r2'` and the uncertainty is vanished. Firings `r3', r4', r5'` originates from `r2` and describe the incorrect output more precisely. The same is true for `r3` and `r6', r7', r8'`.

4 Conclusion and Future Work

In this work we presented a modeling approach and a design environment, that can be used in the early phases of HW-SW codesign. It supports diagnostic design as an integral part of the design process, since in the proposed dataflow model both the functional and error propagation/fault effects information are incorporated.

The prototype of the toolboxes (shaded area in Figure 1) is implemented, and the first credibility studies are executed. As model editor the graphical editor of Ptolemy [7] is used. A graphical, interactive dataflow simulator supports the work of the designer during model construction and verification. For test generation an adaptation of the well known PODEM ([1] algorithm is used. A concurrent fault simulator extracts the input model of integrated diagnostics [8] from the test set of the system. Finally the diagnostic measures can be evaluated by means of the integrated diagnostic tool. Current work incorporates the modeling and analysis of some very complex digital computing systems (evaluation of MEMSY is nearly finished).

Future work involves the integration of the test generator, fault simulator and integrated diagnostics modules into Ptolemy. Theoretical work has to be done on the field of a possible automatic extension of the basic description of the nodes and we want to identify and examine the constraints imposed by the various testing criteria on HW-SW separation.

Acknowledgments

The author wants to express his gratitude to Prof. M. Dal Cin and Mr. W. Hohl, whom hosted him for a period at the University of Erlangen, Germany. The helpful comments of Prof. A. Pataricza are also acknowledged.

References

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

[2] A. Bondavalli and L. Simoncini. Functional Paradigm for Designing Dependable Large-Scale Parallel Computing Systems. In *Proceedings of the International Symposium on Autonomous Decentralized Systems, ISADS '93*, pages 108–114, Kawasaki, Japan, 1993.

[3] G. Boriello, K. Buchenrieder, R. Camposano, E. Lee, R. Waxman, and W. Wolf. Hardware/Software Codesign. *IEEE Design and Test of Computers*, pages 83–91, March 1993.

[4] Gy. Csertán, A. Pataricza, and E. Selényi. Dependability Analysis in HW-SW codesign. In *Proceedings of the IEEE International Computer Performance and Dependability Symposium, IPDS'95*, pages 316–325, April Erlangen, Germany, 1995.

[5] Technical REPORT FUTEG-4/1995. Contract number CP93:9624.

[6] B. Jonsson. A Fully Abstract Trace Model for Dataflow Networks. In *Proceedings of the 16th ACM symposium on POPL*, pages 155–165, Austin, Texas, 1989.

[7] J. Rozenblit and K. Buchenrieder, editors. *Codesign*. IEEE Press, 1995.

[8] W. R. Simpson and J. W. Sheppard. *System Test and Diagnosis*. Kluwer Academic Publishers, 1994.