An Integrated Design Environment for Performance and Dependability Analysis[1]

Robert H. Klenke
Moshe Meyassed
James H. Aylor
Barry Johnson
Ramesh Rao
Anup Ghosh
Department of Electrical Engineering
University of Virginia

Center For Semicustom Integrated Systems
Department of Electrical Engineering
University of Virginia
Charlottesville, Virginia 22903-2442

CSIS Technical Report #970528.0

This is a long version of a paper that appeared in the proceedings of the 1997 Design Automation Conference

1. Support for this work was provided in part by the Defense Advanced Research Projects Agency under contract number F33615-93-C-1313, the Semiconductor Research Corporation under contract number 93-DJ-152, the International Business Machines Corporation under contract number WM-226181, and Union Switch and Signal under contract number A-91846-24

An Integrated Design Environment for Performance and Dependability Analysis

Abstract

This paper presents a integrated design environment that supports the design and analysis of digital systems from initial concept to the final implementation. The environment supports both system level performance and dependability analysis from a common modeling representation. A tool called ADEPT (Advanced Design Environment Prototype Tool) has been developed to implement the environment. ADEPT is based on IEEE 1076 VHDL and uses commercial schematic capture systems as a front end via an EDIF interface. Several examples are presented which demonstrate various aspects of the environment.

1. Introduction

It has been noted by the digital design community that the greatest potential for additional cost and iteration cycle time savings is through improvements in tools and techniques that support the early stages of the design [1]. As shown in Figure 1, decisions made during the initial phases of a product's development cycle determine up to 80% of its total cost. The result is that accurate, fast analysis tools must be available to the designer at the early stages of the design process to help make these decisions. Design alternatives must be effectively evaluated at this level with respect to multiple metrics, such as performance, dependability, and testability. This analysis capability will allow a larger portion of the design space to be explored yielding higher quality as well as lower cost designs.

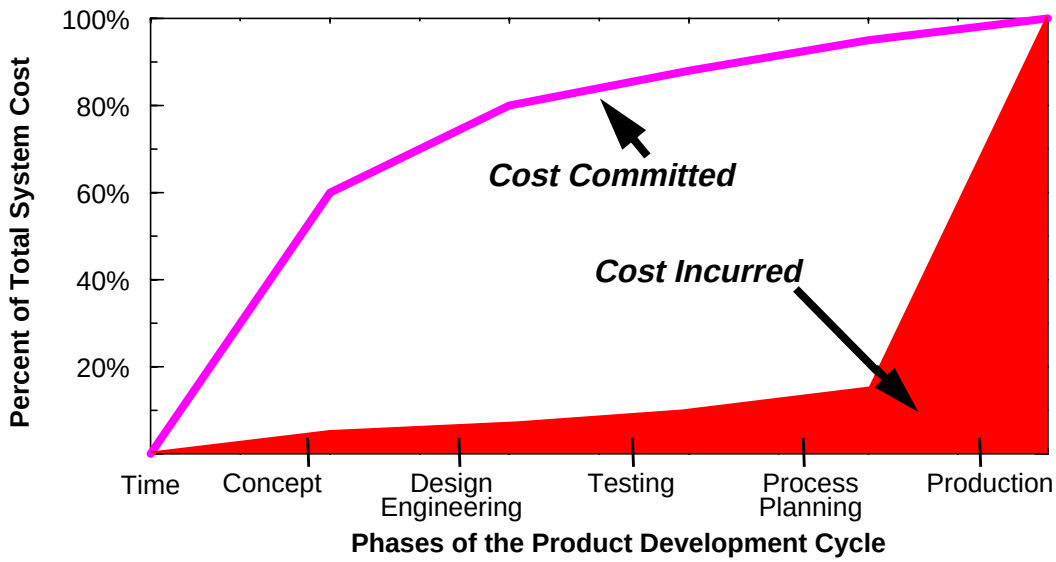


Figure 1. Product costs over the development cycle

There are a number of current tools and techniques that support analysis of these metrics at the system level to varying degrees. A major problem with these tools is that they are not integrated into the engineering design environment in which the system will ultimately be implemented. This problem leads to a major disconnect in the design process where the system level model is developed and analyzed, and then the resulting high level design is specified on paper and thrown "over the wall" for implementation by the engineering design team, as illustrated in Figure 2. As a

result, the engineering design team has to interpret this specification in order to implement the system, which often leads to design errors. They also have to develop their own initial “high level” model from which to begin the design process in a top down manner. Additionally, there is no automated mechanism by which feedback on design assumptions and estimations can be provided to the system design team by the engineering design team.

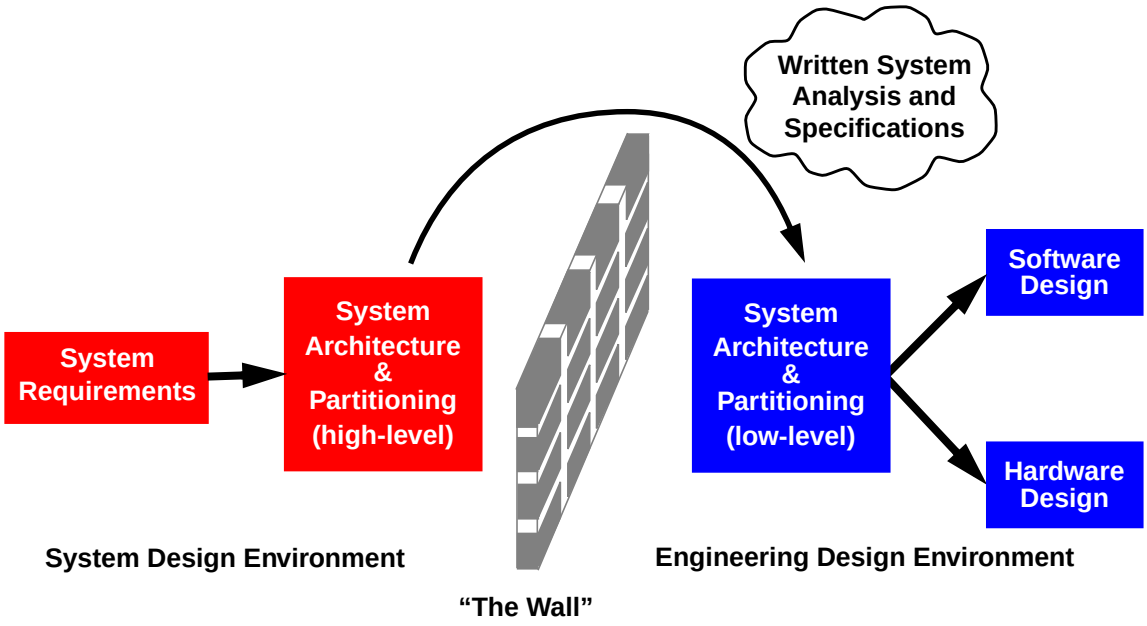


Figure 2. The disconnect between system level design environments and engineering design environments

A further problem with existing system level modeling tools is that they force the designers to represent the system using different modeling paradigms for each type of analysis that is to be performed. For example, even at the system level, multiple models in different representations are needed to analyze performance and dependability. A great deal of extra work must be done to verify that these models accurately represent the same system. This problem is illustrated in Figure 3. All of these problems could be solved to a large degree if a single system level representation could be used to measure multiple metrics and then be used as a starting point for the engineering design process.

This paper presents a unified end-to-end design environment developed at the University of

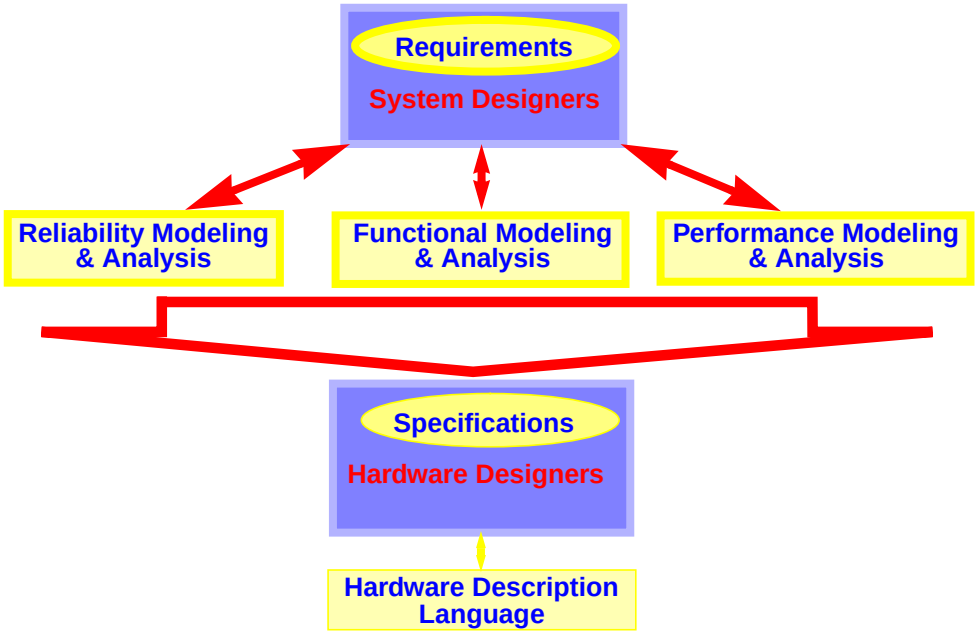


Figure 3. The problem of multiple models required for different analyses

Virginia that attempts to achieve this goal. This environment supports the development of system level models of digital systems that can be analyzed for multiple metrics like performance and dependability, and can then be used as a starting point for the actual implementation. A tool called ADEPT (Advanced Design Environment Prototype Tool) has been developed to implement this environment. ADEPT supports both system level performance and dependability analysis in a common design environment using a collection of predefined library elements. ADEPT also includes the capability to simulate both system level and implementation level (behavioral) models in a common simulation environment. This capability allows the stepwise refinement of system level models into implementation level models.

The remainder of this paper is organized as follows. Section 2 describes the ADEPT environment and set of tools. Section 3 describes the performance modeling capabilities and hybrid modeling capabilities of ADEPT through the use of an example. Section 4 describes the dependability analysis capabilities of ADEPT through additional examples and shows how this analysis can be continued through the design process as the system model is refined. Finally,

Section 5 contains some conclusions.

2. The ADEPT Design Environment

Two approaches to creating a unified design environment are possible. An evolutionary solution is to provide an environment that "translates" data from different models at various points in the design process and creates interfaces for the non-communicating software tools used to develop these models. With this approach, users must be familiar with several modeling languages and tools. Also, analysis of design alternatives is difficult and is likely to be limited by design time constraints.

A revolutionary approach, the one being developed in ADEPT, is to use a single modeling language and mathematical foundation. This approach uses a common modeling language and simulation environment which decreases the need for translators and multiple models, reducing inconsistencies and the probability of errors in translation. Finally, the existence of a mathematical foundation provides an environment for complex system analysis using analytical approaches.

Simulators for hardware description languages accurately and conveniently represent the physical implementation of digital systems at the circuit, logic, register-transfer, and algorithmic levels. By adding a system level modeling capability based on extended Petri Nets and queuing models to the hardware description language, a single design environment can be used from concept to implementation. The environment would also allow for the mixed simulation of both *uninterpreted* (performance) models and *interpreted* (behavioral) models due to the use of a common modeling language.

ADEPT implements an end-to-end unified design environment based upon the use of the VHSIC Hardware Description Language (VHDL), IEEE Std. 1076 [2]. ADEPT supports the integrated performance and dependability analysis of system level models and includes the capability to simulate both uninterpreted and interpreted models in a common simulation environment using a technique called *hybrid modeling*. Hybrid modeling allows the stepwise refinement of system level models into implementation level models. ADEPT also has a mathematical basis in Petri Nets thus providing the capability for analysis through simulation or

analytical approaches [3].

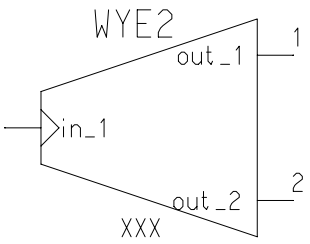
2.1 The ADEPT Modules

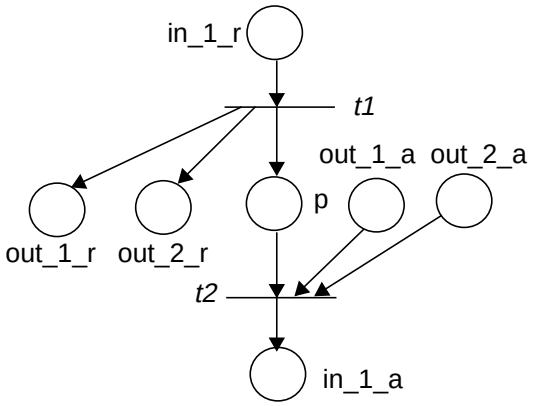
In the ADEPT environment, a system model is constructed by interconnecting a collection of predefined elements called ADEPT modules. The modules model the information flow, both data and control, through a system. Each ADEPT module has a VHDL behavioral description and a corresponding mathematical description in the form of a colored Petri Net (CPN) based on Jensen's CPN model [4]. The modules communicate by exchanging *tokens*, which represent the presence of information, using a fully interlocked, four-state handshaking protocol [5]. The basic ADEPT modules are intended to be building blocks from which useful modeling functionality can be constructed. In addition, custom modules can be developed by the user if required and incorporated into a system model as long as the handshaking protocol is adhered to. Finally, some libraries of application-specific, high-level modeling modules such a Multiprocessor Communications Network Modeling Library [6] have been developed and included in ADEPT.

ADEPT tokens are implemented as a VHDL record structure. In the token, the two most important fields are the *STATUS* field and the *COLOR* field. The STATUS field is used to implement the token passing mechanism; that is, the "handshaking" between the ADEPT modules. The COLOR field is an array of integers that hold user-specified information. Modules are provided which can manipulate the information in the COLOR field.

An example of an ADEPT module is the *Wye*, shown in Figure 4 with its VHDL behavioral description and its underlying CPN representation. This module models a "fork" construct. When a token arrives at in_1, tokens are placed simultaneously at out_1 and out_2. The input token is not acknowledged (consumed) until both output tokens have been acknowledged.

In the Petri Net of Figure 4, the places are shown as circles and the transitions are shown as horizontal lines. The places labeled as "xx_x_r" and "xx_x_a" correspond to "ready" and "acknowledge", respectively. The ready and acknowledge places emulate the handshaking between modules. When a token arrives at the place labeled "in_1_r", the top transition is enabled, and a token is placed in the "out_1_r", "out_2_r", and "p" places. The first two places correspond to a token being placed on the module outputs (out_1 and out_2). Once the output






```
library uvalib;
   use uvalib.uva.all;
   use uvalib.rng.all;

entity wye2 is
   port (in_1 : inout token;
         out_1: inout token;
         out_2: inout token);
end wye2;

architecture ar_wye2 of wye2 is
begin
   pr_wye2 : process (in_1, out_1, out_2)
   begin
      if token_present (in_1)
         and token_removed (out_1)
         and token_removed (out_2) then
         out_1 <= in_1;
         out_2 <= in_1;
      end if;

      if token_acked (out_1) and token_acked (out_2) then
         ack_token (in_1);
         release_token (out_1);
         release_token (out_2);
      end if;

      if token_released (in_1) then
         remove_token (in_1);
      end if;
   end process pr_wye2;
end ar_wye2;
```

Figure 4. Wye module ADEPT symbol, its behavioral VHDL description, and its CPN representation

tokens are acknowledged (corresponding to tokens arriving at the “out_1_a” and “out_2_a” places), the lower transition is enabled, and a token is placed in “in_1_a” (corresponding to the input token being acknowledged). The module is then ready for the next input token. Other modules are modeled similarly. The complete CPN descriptions of each of the ADEPT modules can be found in [7].

The set of basic ADEPT modules is divided into six categories: *control* modules, *color* modules, *delay* modules, *fault* modules, *miscellaneous* parts modules, and *hybrid* modules. The control modules are used to manipulate the flow of tokens in a model. A majority of the control modules have been adapted from Dennis [8]. The *Wye* module described above is an example of a

control module. ADEPT modules in the color and delay categories enable the manipulation of the token color and model temporal aspects of a system, respectively. The fault modules are used to model the presence of faults and errors in a system model. The miscellaneous modules are modules that perform data collection with the ADEPT system. Hybrid modules aid in the construction of hybrid models. A more detailed description of the entire ADEPT module set can be found in [9].

2.2 The ADEPT Tools

The ADEPT system is available on Sun platforms using Mentor Graphics' *Design Architect* as the front end schematic capture system, or on Windows PCs using OrCAD's *Capture* as the front end schematic capture system. The overall architecture of the ADEPT system is shown in Figure 5.

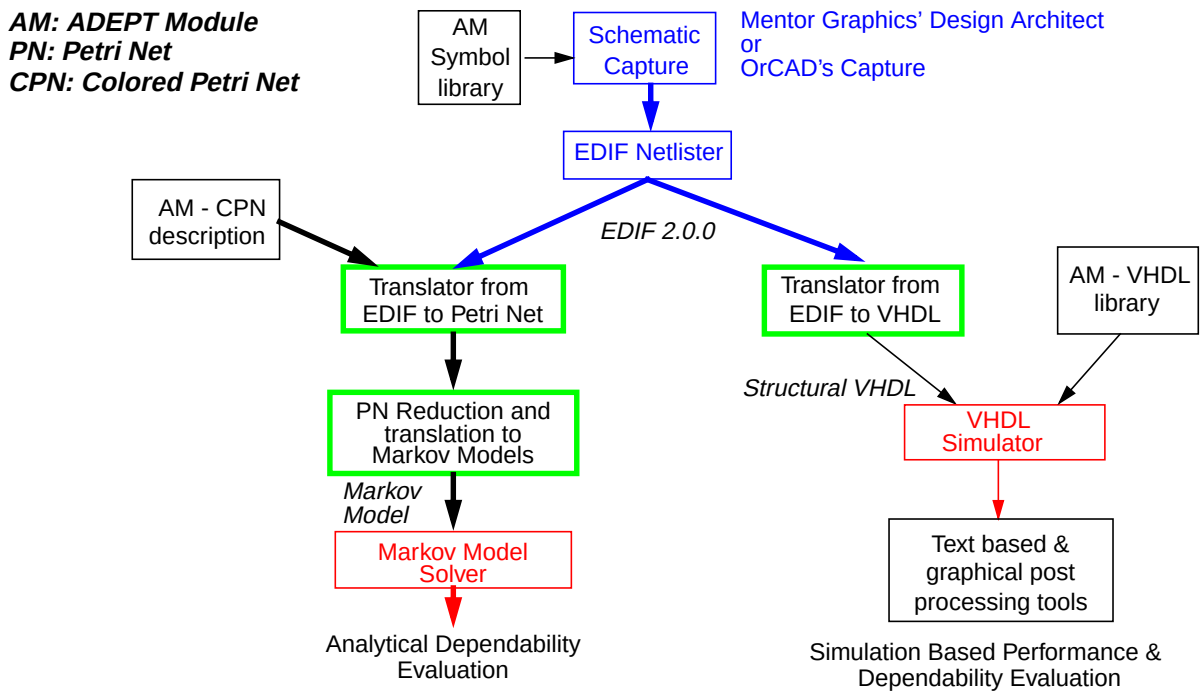


Figure 5. ADEPT design flow

The schematic front end is used to graphically construct the system model from a library of ADEPT module symbols. Once the schematic of the model has been constructed, the schematic capture system's netlist generation capability is used to generate an EDIF (Electronic Design

Interchange Format) 2.0.0 netlist of the model. Once the EDIF netlist of the model is generated, the ADEPT software is used to translate the model into a structural VHDL description consisting of interconnections of ADEPT modules. The user can then simulate the structural VHDL that is generated using the compiled VHDL behavioral descriptions of the ADEPT modules to obtain performance and dependability measures.

In addition to VHDL simulation, a path exists that allows the CPN description of the system model to be constructed from the CPN descriptions of the ADEPT modules. This CPN description can then be translated into a Markov model using well known techniques and then solved using commercial tools to obtain reliability, availability, and safety information.

Figure 6 is an illustration of the construction of a schematic of an ADEPT model using Design Architect. The schematic shown is that of an ADEPT model of a simple three computer system used in the ADEPT tutorial. Most of the elements in this top-level schematic are hierarchical, with separate schematics describing each component. The most primitive elements of the hierarchy are the ADEPT modules.

3. Performance and Hybrid Modeling with ADEPT

This section presents some of the performance analysis capabilities of ADEPT and how the technique of hybrid modeling can be used to gain more accurate performance metrics.

3.1 Performance Modeling

The example used to demonstrate the performance modeling capabilities of ADEPT consists of an FIR digital filter that operates on an incoming stream of data in real-time. The FIR digital filter has to perform a series of multiplications and additions while maintaining the restrictions on their order dictated by data dependencies. The specifications of this system include the required frequency response of the filter and the rate in which the data arrives to the filter. The required frequency response determines the order of the filter, or the number of additions and multiplications required between each arriving data item. The data arrival rate along with the order determines the rate at which additions and multiplications must be performed. Based on these specifications, the designers have to define the system’s architecture, for example, the

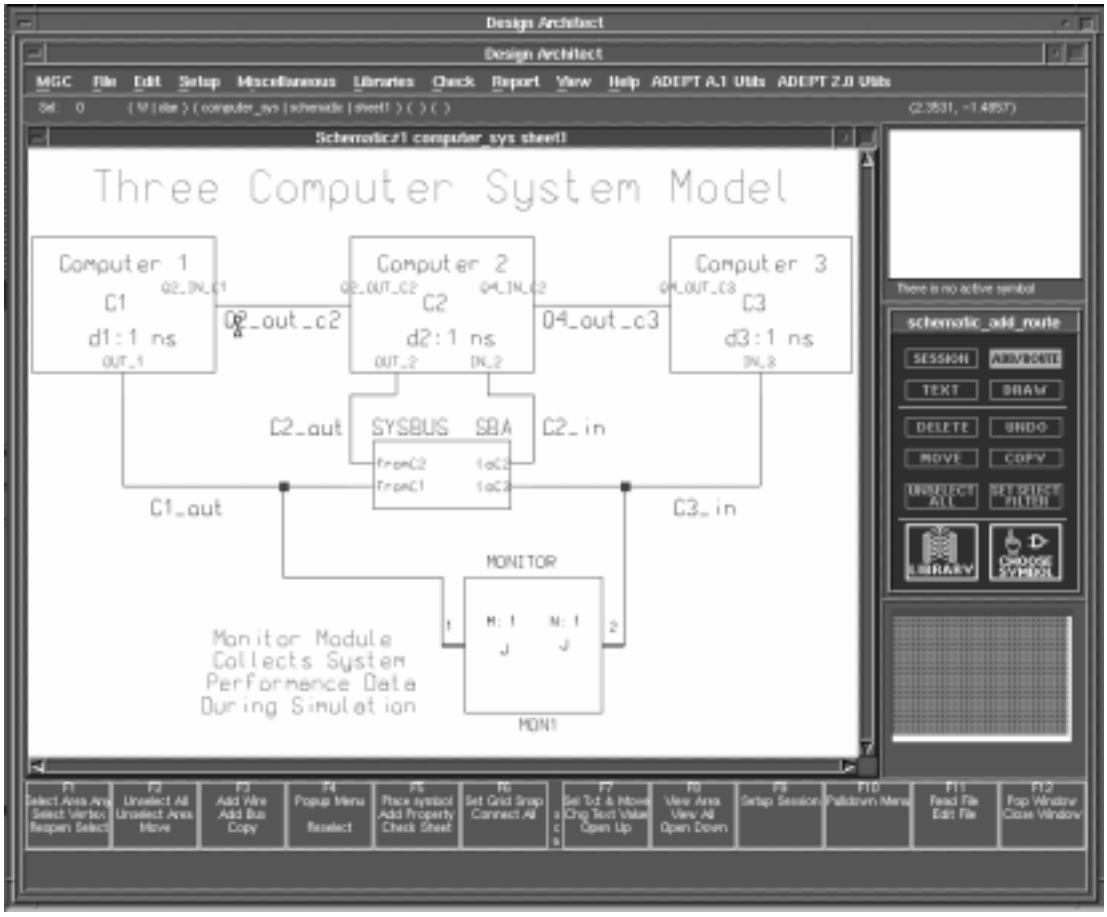


Figure 6. Sample ADEPT schematic (Design Architect)

number and configuration of arithmetic resources to be used.

Figure 7 shows an ADEPT model of one possible configuration of the FIR filter. This model is constructed entirely of ADEPT modules. Some of the elements seen are ADEPT modules (such as *source*, *arbiter* etc.) and some are hierarchical elements built from the ADEPT modules (such as the *resources* and the *fir_cell*). The goal of this model is to analyze the performance of a FIR filter with two adders and two multipliers. At this stage of the design, an estimated delay is associated with each arithmetic resource. The *fir_cell* is responsible for sending the data to the resources while maintaining the data dependencies.

During the simulation of this model, the source generates tokens at a regular rate representing the data stream. Upon receiving this token, the *fir_cell* generates tokens that are sent to the

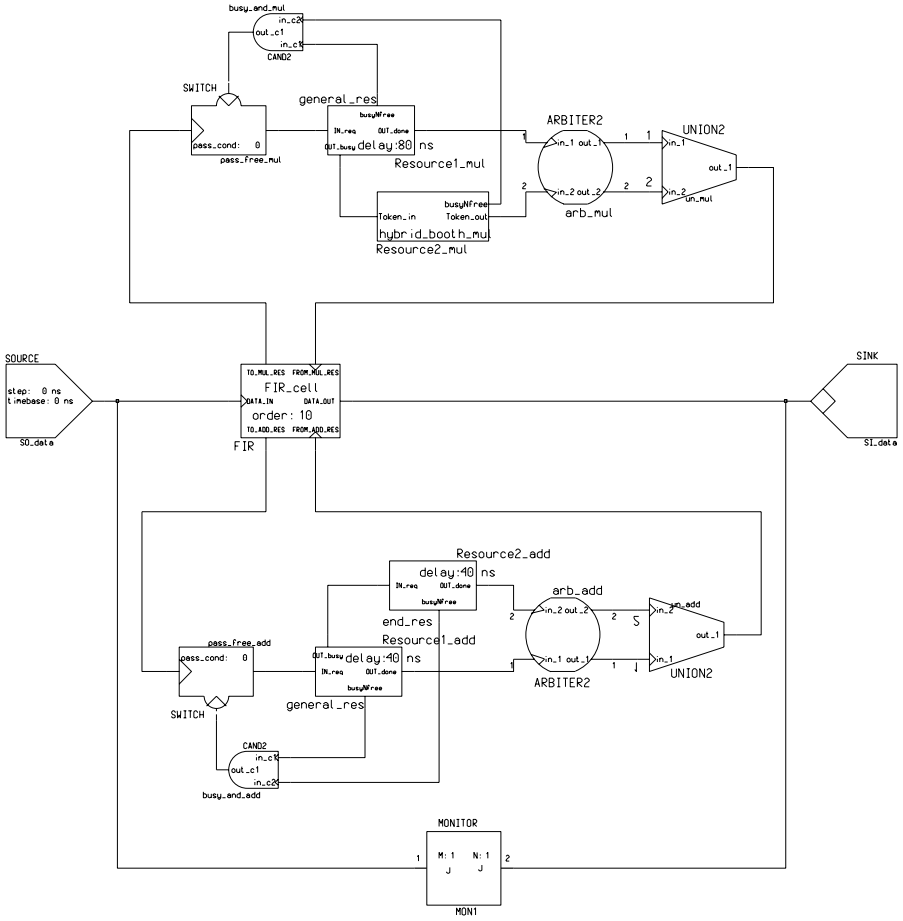


Figure 7. An FIR performance model in ADEPT

resources (multipliers and adders) whenever they are not busy. After delaying each token for a specified time, a resource sends it back to the *fir_cell*, signifying the completion of the desired operation.

By simulating this model it is possible to estimate the effect of different parameters on the performance. The graph in Figure 8 shows the overall system latency as a function of the speed of the multiplier resources for three different filters (order 10, 15 and 20). As expected, for “relatively slow” multipliers (propagation delay of more than 40 ns) the system’s latency is dominated by the multipliers while for “faster” multipliers the system’s latency is dominated by the adders. For this FIR architecture with two multipliers and two adders, the designer can determine the speed required of the multipliers to provide a given data rate and frequency response (filter order). Alternatively, given a specific multiplier implementation, filter order can

be traded off for increased data rates. Finally, it is possible to evaluate different architectures for the system that have different numbers of arithmetic resources. This will require minor modifications to the ADEPT model and resimulation to obtain the new data.

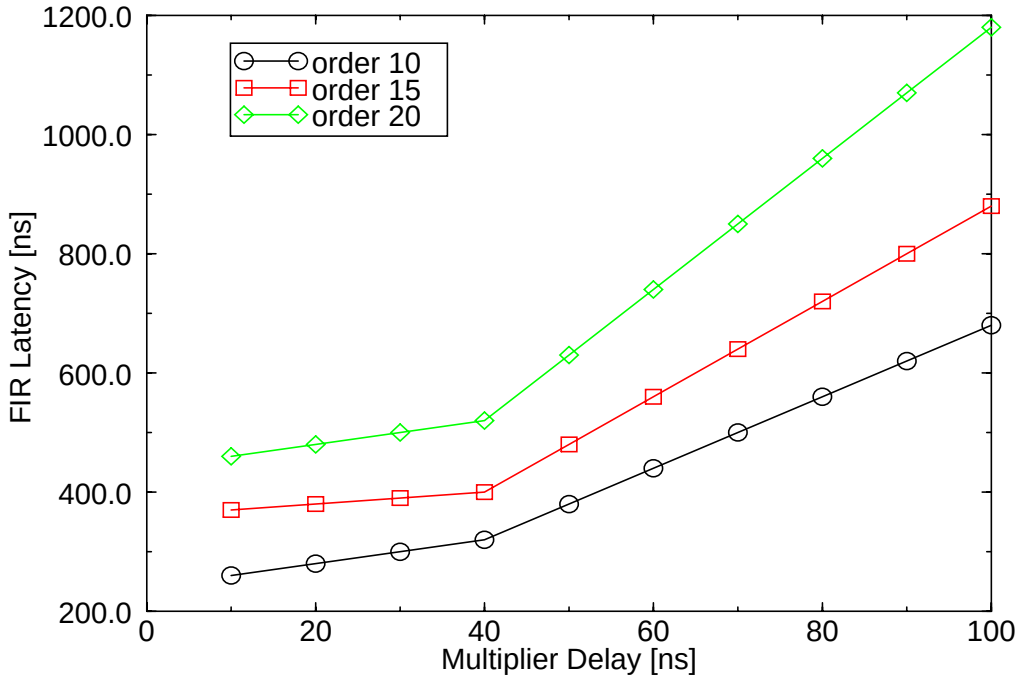


Figure 8. Performance model simulation results - an example

3.2 Hybrid Modeling

Once the initial design space exploration has been done using estimated delays in the performance model, the next step in a typical spiral design process is to implement the portions of the design that have been identified as the critical path to determine if their performance constraints can be met. In ADEPT, this analysis is performed using hybrid modeling.

In a hybrid model a portion of the ADEPT model is replaced by the fully implemented behavioral description of the corresponding component. The result is a model which is composed of an uninterpreted model which manipulates tokens and an interpreted model which manipulates actual signal values (bits, integers, reals, etc.) and has full timing information. This unique hybrid modeling approach is supported by the VHDL implementation of ADEPT.

Since hybrid models mix components that communicate via tokens and components that communicate via values, there is a need for an interface that will handle the flow of information between the two domains. The hybrid interface is divided into two parts, an uninterpreted to interpreted (U/I) part that converts tokens to values, and an interpreted to uninterpreted (I/U) part that converts values to tokens. In addition to value conversion, these interfaces must handle the timing of the two different levels of abstraction during simulation. This portion of the interface function differs significantly depending on the characteristics of the interpreted component and the objectives of the performance model. More detail on the functions of these interfaces and how they are implemented in ADEPT can be found in [10].

The advantage of hybrid modeling and the operation of the hybrid interface are illustrated by the following example. Assume that the performance analysis has shown that In the FIR design, the multiplier is the critical element in meeting performance goals. A multiplier that uses Booth's algorithm was then selected for the implementation of the multiplier, and a behavioral model of the multiplier was developed. A hybrid model using the behavioral description of the multiplier was then constructed. Figure 9 shows the hybrid element (the implemented component and the interface) which replaces the corresponding abstract model of the multiplier. At this stage of the design, the actual data inputs to the multiplier (multiplier and multiplicand) are not known. This lack of information is handled by the hybrid interface in such a way that the maximum or minimum delay through the implemented component can be measured.

The actual method by which this process is performed for sequential interpreted elements involves manipulation of the sequential element's State Transition Graph. The traversal operation is performed based on a given initial state of the sequential interpreted element when a token arrives at the hybrid interface and a given condition on the outputs of the sequential interpreted element that signify the completion of data processing. The result of this traversal operation is the sequence of input values that will drive the state machine along the longest or shortest paths between starting and ending states, depending on whether maximum or minimum delays are desired. Once the proper input sequence is determined, it is applied by the hybrid interface to the sequential interpreted element.

Despite the fact that the difference in level of abstraction between the uninterpreted and

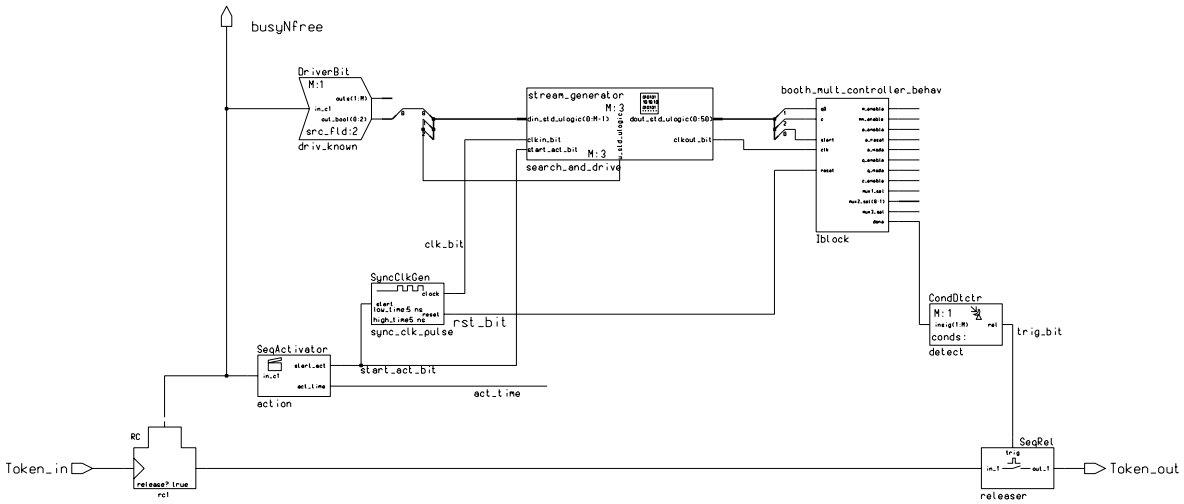


Figure 9. The Booth's multiplier hybrid element

interpreted parts of this hybrid model is quite large, its simulation provides very useful results. The hybrid model simulation provides an upper and lower bound on the performance of this implementation of an FIR. These simulations provide more realistic performance estimation which, in some cases, can suggest modifications in the implementation of the implemented element (multiplier, in this case) or in the architecture of the entire system.

Figure 10 shows some simulation results of the FIR hybrid model described above. This graph shows the maximum and minimum amount of time required to complete one set of FIR computations as a function of the order of the filter. Since the actual delay of the multiply operation is data dependent and the actual data is not known in this stage of performance modeling, the actual system performance will lie somewhere between these two bounds. In a less abstract performance model, more inputs to the interpreted element will be known from the information carried by the tokens, hence narrowing the range between the lower and the upper bounds of the latency.

The nonlinearity of the graphs demonstrates one advantage of hybrid modeling; the maximum and minimum time required to complete a single multiplication by the Booth multiplier can be determined even without developing the hybrid model. However, to find the effect of this multiplier on the latency of the entire system requires the simulation of the hybrid model. This is

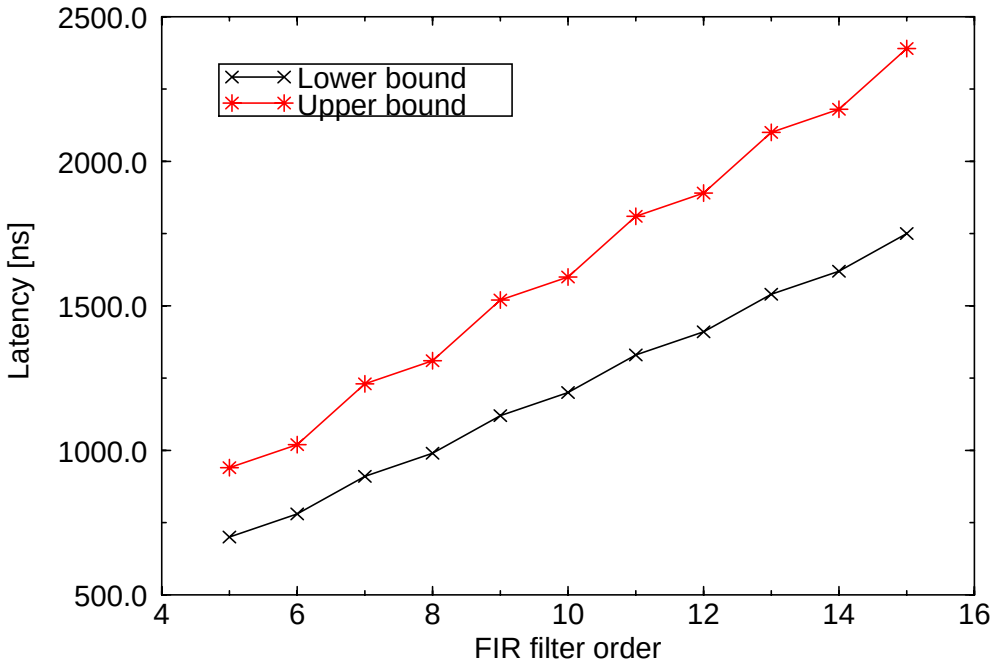


Figure 10. FIR hybrid model: computation time versus filter order

because even in this simple example, there are complex dependencies between delays of the individual components and the overall performance of the system. An additional advantage of hybrid modeling versus off-line simulation of the interpreted element and then back annotating the delay information into the performance model, is that in a hybrid model, the performance model drives the interpreted element. This helps ease the problem of test bench generation for the interpreted element and results in greater accuracy of the overall model because the interpreted element is responding to actual inputs from the overall system instead of contrived examples.

4. Dependability Modeling with ADEPT

ADEPT supports dependability analysis in the same framework and using the same models as the performance analysis. This section presents the capabilities of ADEPT for dependability analysis of system-level models using Petri Net to Markov model conversion, and simulation based dependability analysis for system level and mixed level models using fault simulation.

4.1 High Level Dependability Analysis

System-level dependability analysis is supported in ADEPT by the CPN descriptions of the

ADEPT modules. A system-level ADEPT model can be converted into a CPN description by replacing each module with its CPN description. The CPN model is then reduced using reduction rules developed for dependability analysis [11] and converted to a Markov model using techniques similar to those described in [12]. The Markov model can then be solved to generate dependability metrics using well known techniques and tools.

Dependability analysis using this method is illustrated by an example of a Triple Modular Redundant (TMR) with a Spare system. An ADEPT schematic of the TMR with a Spare system is shown in Figure 11. In this example, it is assumed that there is some form of fault detection in processor P3 that disconnects P3 when it fails and brings processor P4 on-line and that the coverage factor for detecting failures in P3 is 1 (100%), although other values could be used.

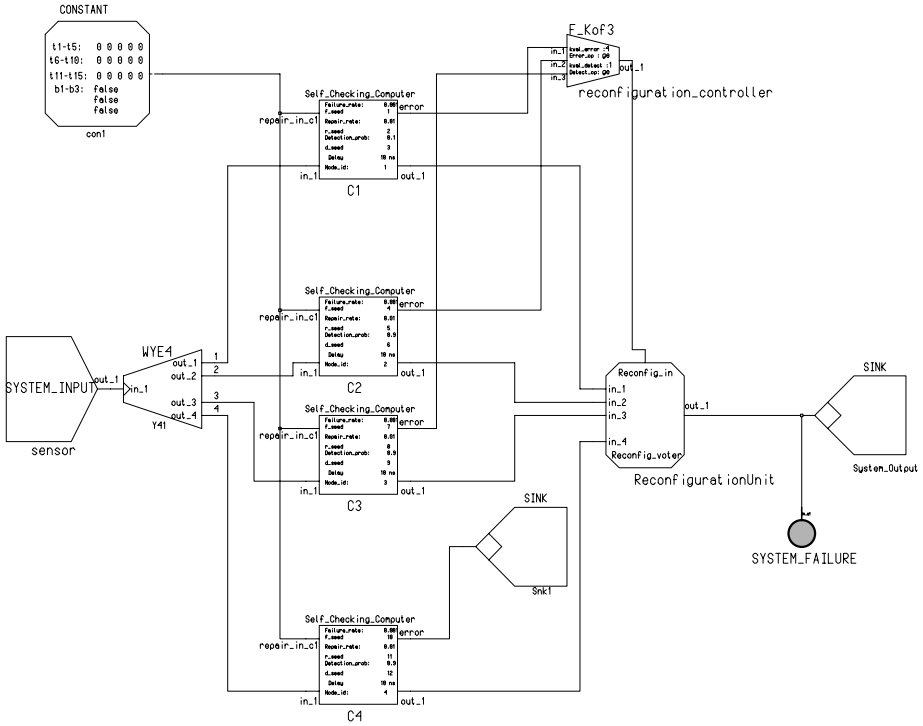
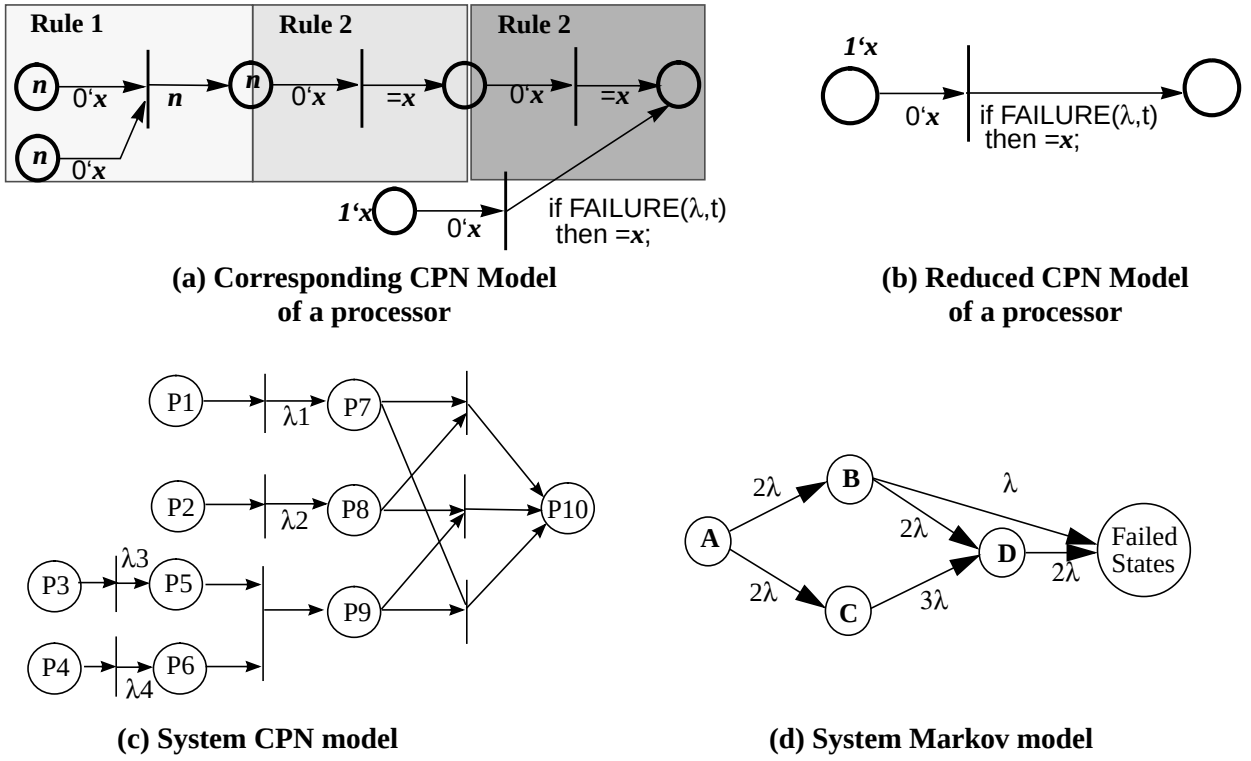


Figure 11. TMR with a spare

The CPN model of each processor module is shown in Figure 12a and is obtained by replacing each ADEPT module by its corresponding reduced CPN definition. Rules used to reduce the CPN in Figure 12a to the CPN in Figure 12b are also illustrated. The remaining components of the system are reduced in a similar fashion and are combined to obtain the reduced CPN

representation of the complete system as shown in Figure 12c. The corresponding Markov model is also shown. The important point here is that the Markov model is constructed from the ADEPT model using automated techniques. The designer does not need to build an additional model in order to gain reliability information. ADEPT also supports reliability analysis using simulation of the system level ADEPT models [13]. Figure 13 shows the results obtained from dependability analysis of the TMR with Spare system in the form of reliability and safety measures versus mission time in hours.



(a) Corresponding CPN Model of a processor

(b) Reduced CPN Model of a processor

(c) System CPN model

(d) System Markov model

Figure 12. TMR with spare model reduction

4.2 Fault simulation in ADEPT

A method of dependability analysis via fault simulation has been implemented in ADEPT. This method of dependability analysis has the benefit of being able to be applied across various levels of abstraction. In this method of dependability analysis, ADEPT models are created using the normal ADEPT tools. A separate tool then adds a new process to the VHDL description generated by ADEPT that performs fault injection during simulation. A VHDL bus resolution

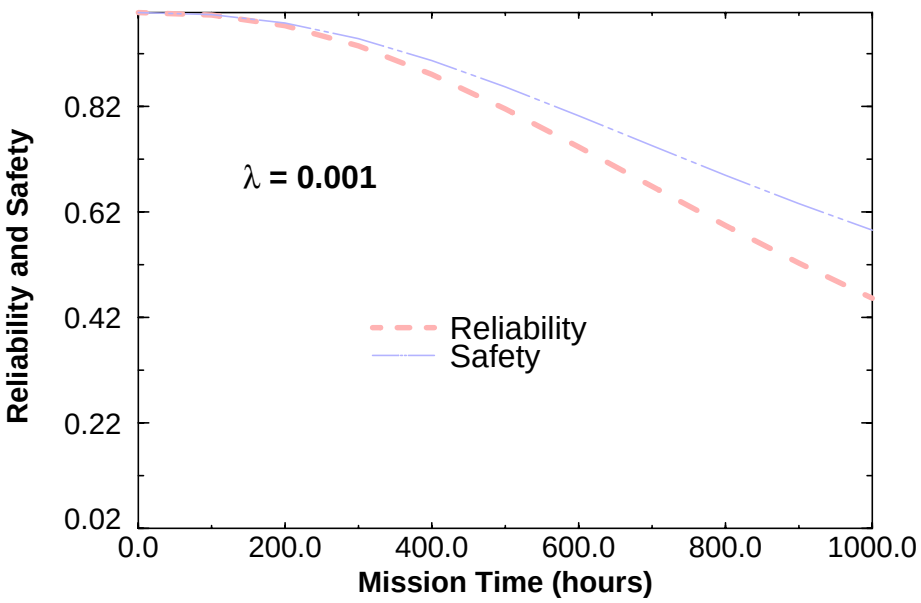


Figure 13. TMR with a spare dependability analysis results

function (BRF) applies the fault from the fault injection process to the data field of the target signal according to the fault operation. The fault injection process is illustrated in Figure 14.

By performing fault injection in VHDL using signal corruption, fault simulation and dependability evaluation can be performed throughout the design process as the ADEPT model is refined to lower levels of abstraction. The fault operations available for fault injection vary with the level of design abstraction as shown in Table 1.

Table 1 Supported Fault Operations in ADEPT

Abstraction Level	Data Types	Fault Operations
Architectural	token, integer, boolean	info loss, info creation, stuck-at, additive, multiplicative, logical
Algorithmic	token, integer, boolean, real, bit	info loss, info creation, stuck-at, additive, multiplicative, logical
Functional Block	qsim_state, bit	stuck-at, logical
Logic Level	qsim_state, bit	stuck-at, logical

Dependability analysis in ADEPT using fault simulation will be illustrated via an example. The system presented for analysis is a watchdog monitor card that is embedded within a real-time distributed system used for wayside train control. Train control functions at the wayside include the switching and signaling of trains on railways. The watchdog monitor card is responsible for assuring the safety of the distributed system. The functions of the watchdog monitor have been synthesized in seven Actel Field Programmable Gate Array (FPGA) chips.

The function of the watchdog monitor is to concurrently check the execution of the safety-critical application within real-time constraints. The watchdog checks the input and output data from the application processor using the method of signature analysis [14]. Signature

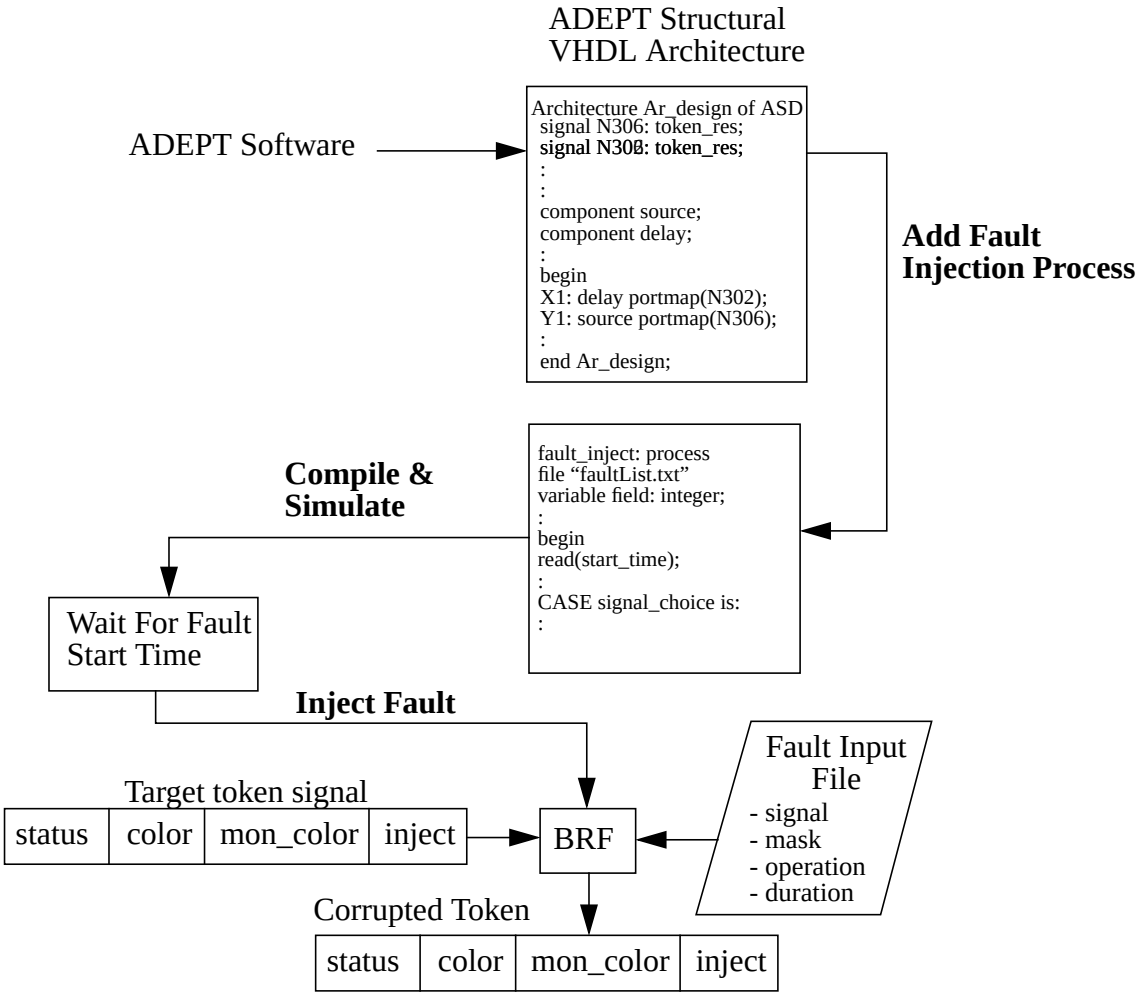


Figure 14. Performing fault simulation in ADEPT

analysis uses Linear Compacting Finite State Machines (L-CFSMs) in order to compact a data stream into a single word that is typically much smaller than the size of the data stream. The signature is checked against a pre-computed fault-free signature to ascertain whether or not the correct data stream was produced.

The watchdog monitor card is embedded within a distributed computer system that is executing a safety-critical train control application. The distributed system is primarily composed of Commercial Off-The-Shelf (COTS) hardware components and custom-developed software. This system is an example of a dependable system built from possibly unreliable COTS components. The COTS components are assumed to be unreliable because they were not built specifically for highly-dependable applications. COTS components are more often being used in highly-dependable applications in order to minimize the amount of custom in-house development — which can significantly reduce costs in system development. In order to meet ultra-high dependability requirements, a watchdog monitor is custom-developed in-house to check the correct execution of the safety-critical application. The design methodology illustrated by this example is a highly dependable application executed by a system composed primarily of COTS components with a sole custom-designed card responsible for system safety.

A model of the watchdog monitor was created at the algorithmic level of design abstraction in order to simulate its functions early in the design process. The algorithmic-level design implements the functions of the watchdog monitor without tying the design to a particular hardware or software implementation. The algorithmic-level design of the watchdog monitor is shown in Figure 15. The main components of the watchdog monitor card are: two L-CFSMs (*MISR64* and *MISR63*), a codeword checker (*CHECK(127,64)*), and a signature comparator (*sigComparatorv2*). Fault simulations were executed on this model to test the response of the design to injected faults. Each signal pictured in the design of Figure 15 was injected with faults in over 450 fault simulation experiments. By examining the response of the simulation to the injected faults, knowledge of which faults affect the output of the watchdog monitor is gained.

The fault simulations corrupted the input and output signals for every functional block in the algorithmic-level watchdog checker. The fault operations used in the simulations spanned the range of fault operations available at the algorithmic level (see Table 1). For these experiments, all

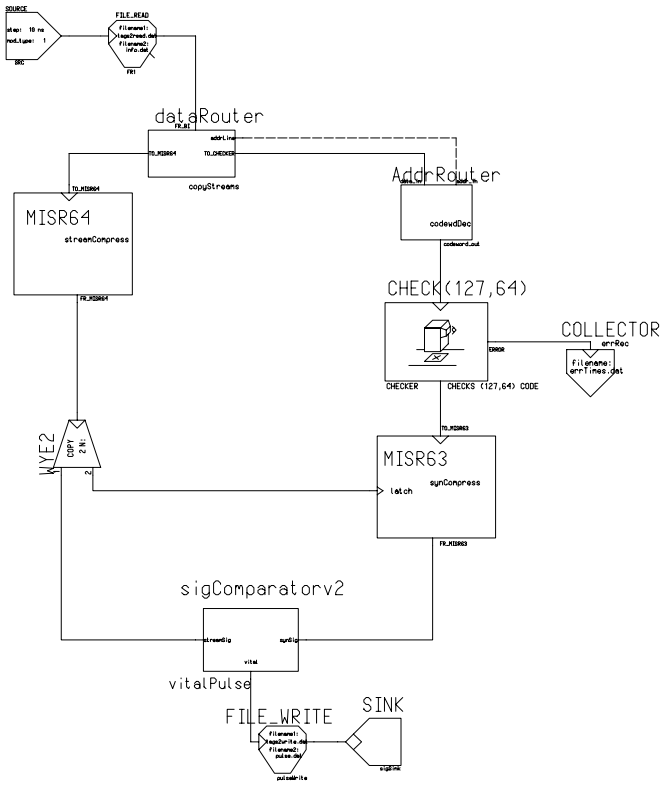


Figure 15. Algorithmic-level design of watchdog monitor card

fields that were corrupted were of type integer with the exception of the status field. Integer fault values were selected from a range of large negative integers to large positive integers. Status field faults consisted of sourcing or sinking faults. Figure 16 shows the results from fault simulations of the algorithmic-level watchdog monitor. The horizontal axis depicts the fault operations that were used in the fault simulations. The fault operations correspond to the types of operations available for token type signals in ADEPT. Stuck-at faults cause the value carried by the signal to be “stuck-at” some particular constant. The additive, multiplicative, and logical fault operations perform their respective operations using the unperturbed value carried by the signal and the fault mask as operands to the fault operation. The result of the fault operation replaces the unperturbed value carried by the signal. Sinking faults set the status of the *token* signals to “removed”, thereby removing the presence of the data. Sourcing faults set the status of *token* signals to “present”, thereby indicating presence of data on the signal.

The results from the fault simulations, indicated by the vertical axis of Figure 16, show the

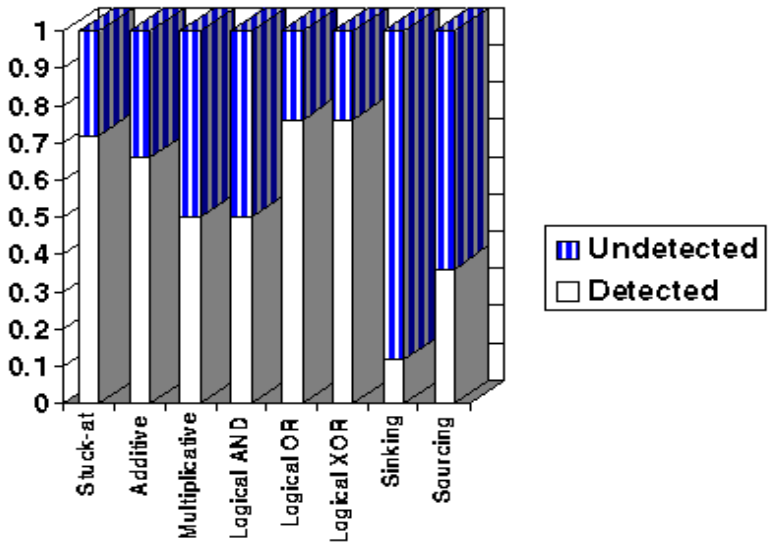


Figure 16. Algorithmic-level watchdog fault simulation results

percentage of faults that were detected and the proportion of faults that remained undetected. An undetected fault indicates that the injected fault was not identified by the built-in detection mechanisms. There may be several possible reasons for why injected faults do not result in detectable errors. First, an injected fault may not actually corrupt the data state of the system. For example, setting a stuck-at-0 fault on a signal whose unperturbed value is always 0 will not cause an error in the output. Second, faults can remain latent for an extended period of time. For example, corrupting a memory location may not effect the data state of the system until this memory state is read and used in processing. Third, some injected faults are masked by data processing operations. For example, corrupted data may be masked by other inputs to components depending on the component function and its input values.

Functional descriptions of the watchdog components were then developed and incorporated into the ADEPT model using hybrid modeling techniques as shown in Figure 17. The watchdog interface module in Figure 17 accepts signals of type *token* on its input and drives *qsim_state* signals on its output. The *token* type signals carry the data stream from the application processor to be checked by the watchdog monitor. Fault simulations of this model can perturb both the ADEPT *token* input signals and the internal *qsim_state* signals of the watchdog monitor.

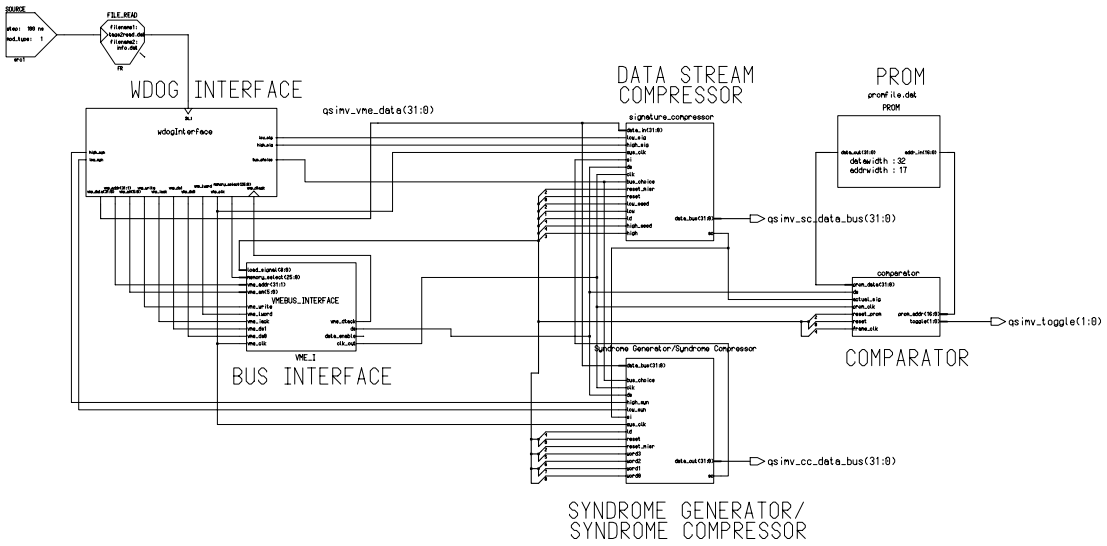


Figure 17. Functional-block-level design of watchdog

The results from injecting faults in the internal design of the functional block level watchdog monitor are shown in Figure 18. The internal signals used in the functional block level watchdog are of type *qsim_state*. Table 1 shows that the supported fault operations for this type are stuck-at-0, stuck-at-1, and the logical AND, OR, and XOR operations. Since logical AND-with-0 and logical OR-with-1 are identical to stuck-at-0 and stuck-at-1 faults, respectively, these two logical fault operations were excluded in favor of using the stuck-at fault operations. In addition, logical XOR-with-1 fault injections were used in order to “flip” the bit of the *qsim_state* signal being injected. The results in Figure 18 indicate that the majority of the injected stuck-at faults remained undetected at the output. In contrast, the majority of the logical XOR-with-1 injected faults were detected. It can be concluded that this circuit is reasonably good at “hiding” stuck-at faults, while more likely to reveal flipped bit faults. It is important to recognize, however, that different results may be obtained using different input streams and longer simulation times.

Finally, the comparator component of the functional block level watchdog was synthesized to the logic level in VHDL. Fault simulation can be performed at the logic level to validate the actual implementation of the fault detection mechanisms. The design of the watchdog monitor remains identical as shown in Figure 17 with the exception that the comparator was replaced with its logic level implementation. The design now contains components modeled at multiple levels of design

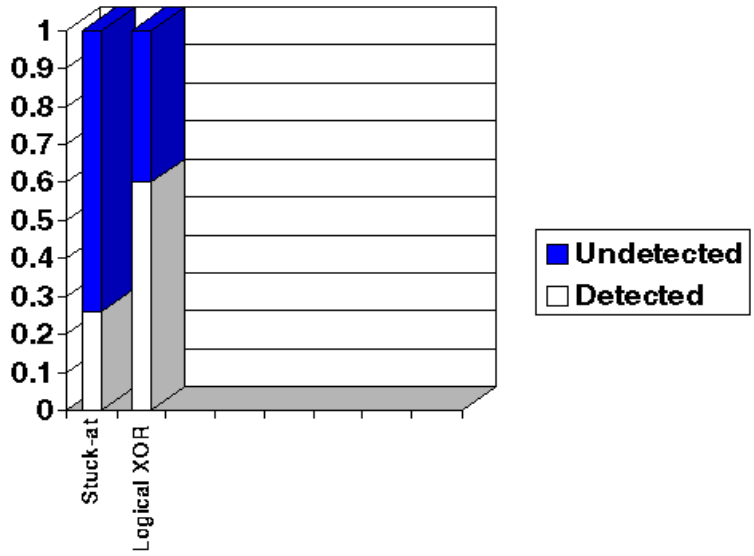


Figure 18. Functional block-level watchdog monitor fault simulation results

abstraction including the watchdog interface at the algorithmic level; the MISRs and syndrome generators at the functional block level, and the comparator at the logic level. Since the models are all VHDL, the design can be simulated seamlessly. The fault simulations of this design were re-run under identical input conditions and similar fault injections. The results from injecting faults in the internal *qsim_state* signals of the design are shown in Figure 19. These results show no substantial statistical difference from the fault simulations at the functional block level. It can be seen that the fault coverage remains relatively unchanged even after synthesizing the comparator component at the logic level.

5. Conclusions

An integrated design environment called ADEPT that supports performance and dependability modeling at the system level has been developed. In addition ADEPT supports the modeling and analysis of systems at mixed levels of abstraction. ADEPT models may contain elements modeled both uninterpreted and interpreted levels. Performance and dependability analysis can be performed on these mixed level models as well. This allows true step-wise refinement of system level models down to an implementation level. A complete set of tools has been developed to help automate the construction and analysis of ADEPT models. Work is on

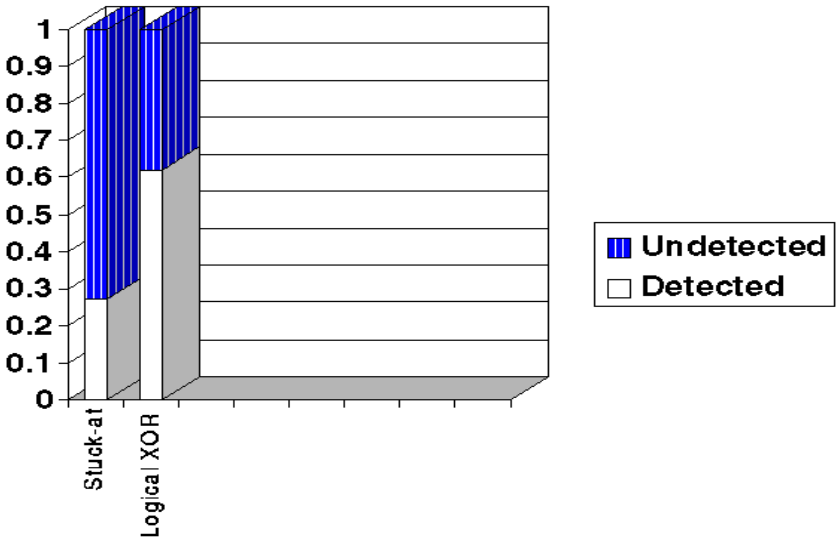


Figure 19. Logic-level watchdog monitor fault simulation results

going to extend the ADEPT environment into the areas of operational specifications and hardware/software codesign.

6. References

[1] ASIC & EDA, January 1993

[2] IEEE, *"IEEE Standard VHDL Language Reference Manual,"* New York, NY, IEEE Std. 1076-1993, June 6, 1994.

[3] J. H. Aylor, R. Waxman, B. W. Johnson, R. D. Williams, "The Integration of Performance and Functional Modeling in VHDL," in *Performance and Fault Modeling with VHDL*, J. M. Schoen (Ed.), Prentice-Hall, Englewood Cliffs, NJ, 1992, pp. 22-145.

[4] K. Jensen, "Colored Petri Nets: A high level language for system design and analysis," in *High-level Petri Nets: Theory and application*, K. Jensen and G. Rozenberg (Eds.), Berlin: Springer-Verlag, 1991, pp. 44-119.

[5] F. T. Hady, *A Methodology for the Uninterpreted Modeling of Digital Systems in VHDL*, Master's Thesis, Dept. of Electrical Engineering, University of Virginia, January 1989.

[6] A. P. Voss, "Analysis and Enhancements of the ADEPT Environment," Master of Science (Electrical Engineering) Thesis, University of Virginia, May 1996.

[7] G. Swaminathan, R. Rao, J. H. Aylor, and B. W. Johnson, *Colored Petri Net descriptions for the UVA primitive modules,* CSIS Technical Report 920922.0, University of Virginia, September 1992.

[8] J. B. Dennis, “Modular, Asynchronous Control Structure for a High Performance Processor,” *ACM Conference Record, Project MAC,* Massachusetts, 1970, pp. 55-80.

[9] *ADEPT A.1 Library Reference Manual*, CSIS Technical report 960625.0, University of Virginia, June 6, 1996.

[10] M. Meyassed, R. M. McGraw, J. H. Aylor, R. H. Klenke, R. D. Williams, F. Rose, and J. Shackleton, “A Framework for the Development of Hybrid Models,” Proceedings of the 2nd Annual RASSP Conference, July, 1995, pp. 147-154.

[11] R. Rao, G. Swaminathan, B. W. Johnson, and J. H. Aylor, “Synthesis of Reliability Models from Behavioral-Performance Models,” Proceedings of the 1994 Reliability and Maintainability, Symposium (RAMS), January 1994, pp. 292-297.

[12] M. K. Molloy, *Performance Analysis Using Stochastic Petri Nets*, IEEE Transactions on Computers, Vol. 31, No. 9, Sept. 1982, pp. 913-917.

[13] E. D. Cutright and B. W. Johnson, “A Simulation-based Approach to Integrated Performance and Reliability Modeling using VHDL,” Proceedings of the 1994 Reliability and Maintainability Symposium (RAMS), January 1994, pp. 402-408.

[14] P.H. Bardell, W.H. McAnney, J. Savir, *Built-in Test for VLSI: Pseudorandom Techniques*, John Wiley & Sons, 1987.