Model-Based Diagnosis - An ASML Case Study

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Anup Barve

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
The Netherlands
Model-Based Diagnosis - An ASML Case Study

by Anup Barve

Abstract

With the increasing complexity of the embedded systems, there is an ever-present need of securing and maintaining their safe and reliable operation. Fault diagnosis is an integral part of maintaining such systems. Model Based Diagnosis (MBD) is a promising technique that considerably speeds up the process of fault finding and increases the accuracy of diagnosis.

Although MBD has an enormous industrial potential, the number of industrial applications that actually make use of MBD are still negligible. One of the reasons is the lack of an environment that implements the diagnosis techniques. An important impediment is the fact that system models are required as an input to the MBD process.

The work in this thesis aims at deriving (semi-)automatic fault diagnostic system models from the existing electrical design specifications (Electrical Layouts) for a sub-system in ASML lithography machine, applying these models to system specific realtime data and finding faulty components using a diagnosis engine. Additionally, we aim at evaluating the diagnostic performance by comparing the output from the diagnosis engine with the diagnosis with inserted health of the simulated system.

Keywords : Model-based Diagnosis, Electrical Layouts, Diagnosis Performance

Faculty : Computer Engineering
Department : Software Engineering

Committee Members :

Advisor: Dr. Arjan van Gemund
Member: Dr. Koen Langendoen
Member: Dr. Arie van Deursen
Member: Ir. Jurryt Pietersma
Member: Ir. Frans Ververs
Dedicated to hard work and sincerity!
Contents

List of Figures vii
List of Tables ix
Acknowledgements xi

1 Introduction 1
  1.1 Model-Based Diagnosis ........................................ 1
  1.2 Lydia ...................................................... 2
  1.3 Electrical Layouts .......................................... 3
  1.4 Thesis Contributions ....................................... 3
  1.5 Thesis Outline ............................................. 4

2 Model-Based Diagnosis and Lydia 5
  2.1 Model-Based Diagnosis ....................................... 5
    2.1.1 Model-Based Diagnosis Theory .......................... 5
    2.1.2 Model-Based Diagnosis Approach ...................... 6
  2.2 Lydia ...................................................... 7
    2.2.1 Lydia Syntax .......................................... 7
    2.2.2 Lydia Simulator ....................................... 10
    2.2.3 Lydia Diagnosis Engine ................................. 10

3 ASML lithography Machine - E-PINS Case Study 13
  3.1 Introduction to ASML lithography Machine .................. 13
  3.2 Wafer Exchange Process .................................... 14
  3.3 Hardware Description ....................................... 14
  3.4 E-PIN Safety Logic ......................................... 15
  3.5 Extracting Data from Board Dumps ........................ 16
  3.6 Why Diagnose with Lydia Model? ............................ 17
  3.7 Why (Semi-)Automate Lydia Model Generation? ............ 17
    3.7.1 E-pin Lydia Model Inferred by Human Expert .......... 18
    3.7.2 Model Details ......................................... 20
    3.7.3 Diagnosis ............................................. 20

4 LYDIA Model Generation from Electrical Layouts 23
  4.1 Introduction ............................................... 23
  4.2 E-layout Structure ......................................... 23
  4.3 E-Layout File Format ...................................... 25
  4.4 Lydia Model Generation Process ............................ 27
    4.4.1 Introduction .......................................... 27
List of Figures

1.1 Model-Based Diagnosis ................................................. 2
1.2 Diagnosis of a System ................................................. 3

2.1 Diagnosis Approach ................................................... 6
2.2 Polycell ................................................................. 8
2.3 Simulation and Diagnosis .............................................. 10

3.1 Hardware Infrastructure ............................................... 14
3.2 E-pin Hardware Schematic .......................................... 16
3.3 Current ASML Diagnosis Method Vs. Diagnosis with Lydia .... 18

4.1 E-layout Anatomy ..................................................... 24
4.2 Lydia Model Extraction from E-layout Design ..................... 27
4.3 E-pin E-layout Representation in Viewdraw ....................... 33

5.1 Accuracy Model ....................................................... 37
5.2 Accuracy Definition 1 - Polycell (Accuracy Vs. Entropy) ....... 41
5.3 Accuracy Definition 1 - Polycell (Accuracy Vs. Entropy) ....... 42
5.4 Accuracy Definition 1 - Polycell (Accuracy Vs. Entropy) ....... 43
5.5 Accuracy Definition 2 - Polycell (Accuracy Vs. Entropy) ....... 43
5.6 Accuracy Definition 2 - Polycell (Accuracy Vs. Entropy) ....... 44
5.7 Accuracy Definition 2 - Polycell (Accuracy Vs. Entropy) ....... 44
5.8 Accuracy Definition 1 - E-pin (Accuracy Vs. Entropy) .......... 45
5.9 Accuracy Definition 1 - E-pin (Accuracy Vs. Entropy) .......... 45
5.10 Accuracy Definition 1 - E-pin (Accuracy Vs. Entropy) .......... 46
5.11 Accuracy Definition 2 - E-pin (Accuracy Vs. Entropy) .......... 47
5.12 Accuracy Definition 2 - E-pin (Accuracy Vs. Entropy) .......... 47
5.13 Accuracy Definition 2 - E-pin (Accuracy Vs. Entropy) .......... 48
5.14 Accuracy Definition 1 - E-pins (Accuracy Vs. Relative Observability) .... 48
5.15 Accuracy Definition 1 - E-pins (Accuracy Vs. Relative Observability) .... 49
List of Tables

3.1 Sensors and Corresponding Register Bits . . . . . . . . . . . . . . . . . . . 17
4.1 Intermediate Format after Pass 1 . . . . . . . . . . . . . . . . . . . . . . . 28
Many thanks to Dr. Arjan van Gemund for being my thesis advisor and for the valuable guidance. Grateful acknowledgment for the valuable suggestions and help from Dr. Andre Bos, my advisor at Science & Technology BV. Also thanks to Jurryt Pietersma for guiding me while working at ASML campus.

Last but not the least, thanks to my family, back home in India for supporting me throughout the thesis period.

Anup Barve
Delft, The Netherlands
August 5, 2005
Monitoring the state of an industrial process plays a significant role in embedded system maintenance. Continuous polling for a healthy system state is required in dynamic systems. If the system is very complex, the task of manually detecting the faults in the system becomes laborious. Hence, a self-automated diagnostic system is required to ensure timely fault detection.

The majority of automated diagnostic systems use expert systems that contain the combinations of all possible faulty component behaviors, in order to locate the faulty components. However, the acquisition of necessary diagnostic knowledge for such expert systems is a key bottleneck. Model-based diagnosis is seen as an emerging solution to overcome this bottleneck, which uses the models that contain internal structure and behavior of the system. These models and sensor observations are fed to the Artificial Intelligence algorithms that operate as diagnosis engines for fault diagnosis.

Manually developing the models from system specific data is a cumbersome job, which should be done by experts. This disadvantage prevents the companies from investing time and effort in modeling.

This research work shows that it is possible to (semi-)automatically extract models from ASML lithography machine specific design, which is in the form of electrical layouts. We will detect faults in the system using these models and model-based diagnosis techniques. Furthermore, we will evaluate the accuracy of the fault diagnosis. Thus we aim at presenting the potential of model-based diagnosis in industrial applications by solving a case study in the electrical engineering domain.

1.1 Model-Based Diagnosis

In this section we summarize the concept of model-based diagnosis. We aim at building up a base for the reader to understand the problem statement by describing sufficient theory behind model-based diagnosis. Detailed description of model-based diagnosis concepts will be comprehensively dealt within the next chapter.

Model-based diagnosis is a process of finding faults (fault locations) in a system on the basis of differences in the expected and the actual behavior of the system [6]. The expected behavior of the system is derived from the model, while the actual behavior is extracted from the real time observations from the system. It is assumed that the expected behavior that is predicted by the model of the system is always correct. Thus, if there is a discrepancy between the expected behavior and the actual observed behavior, we conclude that the system is behaving in a faulty way. Figure 1.1 depicts the diagnostic process.

The comparison between the expected behaviour and the observed behavior should be carried out in such a way that the faulty components are reasoned out on the basis
of the discrepancy. Thus, the output diagnosis would contain a diagnosis array with a list of most probable faulty components.

1.2 Lydia

Throughout the thesis, we use the Lydia language for modeling purpose. Lydia is an acronym for 'Language for sYstems DIAgnosis' [11]. It is a functional, model-based systems language. Lydia can be used to model complex systems in such a way that their behavior can be simulated. Furthermore, this behavior can be analyzed for detecting faults in the system.

In simulation mode, the Lydia model can be compiled to a simulator in Lydia software package. The inputs to the model and health parameters (health variables representing the system components) can be fed to the simulator, which are used by the simulator to give corresponding outputs for the simulated system. Thus, the simulator basically solves the equations for outputs. We need the simulator to capitulate the behavior of the system, since it is often difficult to access the real time data of the system. Although simulation is not a substitute for the real data, simulation is used for model verification and validation. Additionally, simulation makes fault injection easier.

In diagnosis mode, the Lydia model is diagnosed for the existing faults in the system and the fault causes are extracted in terms of the health parameters of the system components. The a-priori probabilities of the health parameters for system components are included in the model and are used as a search heuristic by the diagnosis engine.

We explain detailed Lydia syntax and semantics in Chapter 2.
1.3 Electrical Layouts

Electrical layouts (E-Layouts) are the electrical drawings that represent a complete functional overview of all the electrical functions of a system. E-Layouts ensure that the electrical infrastructure of the subsystem and its modules, is designed, prototyped and qualified in a controlled way.

ASML uses a graphical tool viz.Viewdraw for design capture and analysis of E-Layouts for various systems in the lithography machine. Each E-Layout has two views viz.graphical view and textual view. The textual view incorporates the information about the E-Layout in an ASCI format. We aim at developing a LYDIA model framework by extracting this information, which is explained in Chapter 5.

1.4 Thesis Contributions

Figure 1.2 shows a two-way diagnosis approach for a system. In the more traditional way of diagnosis, diagnosis is carried out using the board dump files and matching the fault symptoms with the look up table entries. The board dumps are recorded from the system specific software that is used to operate the system.

In this thesis we aim at presenting the second diagnosis approach, a complete sequence of model-based diagnosis methodology for a case study associated with a sub-system in an ASML lithography machine, in order to show applicability of MBD in the industrial domain. The sequence includes model extraction from system specific data, fault diagnosis and performance evaluation of the diagnosis as shown in the thesis domain in the figure 1.2.
The main contributions of the thesis can be summarized as follows.

- We develop a tool to (semi-)automatically extract Lydia models from the textual view of the Electrical Layouts that are designed in the Viewdraw software. The tool delivers a framework of the model along with basic behavior of the system. In the process, we develop the grammar for the text files of the Electrical layouts.

- We study and solve a case study that is related to a sub-system in ASML lithography machine. The Lydia model framework is extracted from the Electrical layout of the sub-system using the above mentioned model extraction tool. Furthermore, the model is manually completed with additional relevant behavior and is diagnosed for faults. We focus on a specific error detection in the system. The details of this error (E-PIN error) are presented in Chapter 4.

- We evaluate the performance of diagnosis for different accuracy definitions by developing appropriate experimental environment. In addition, we address the issue of sensor placement by measuring the accuracy of the diagnosis as a function of observability (number of system observables fed to the diagnosis engine).

1.5 Thesis Outline

The thesis is organized as follows.

Chapter 2 describes the theory behind model-based diagnosis. It also contains the description of the Lydia modeling language and available Lydia tools like the Lydia simulator and the Lydia diagnosis engine.

Chapter 3 introduces the ASML lithography machine and presents a potential case study (E-PIN case) for model-based diagnosis. The goal of this chapter is to provide comprehensive details related to the lithography machine that will be related to the the Lydia E-PIN model which is further used for accuracy calculations in Chapter 5.

In Chapter 4, we describe the Lydia model generation process from Electrical Layouts. We present the E-Layout structure and explain the file format used in the electrical designs. Furthermore, we will discuss the experimental setup that is used to extract the Lydia models and demonstrate its working by extracting a Lydia model for the E-PIN case study.

Chapter 5 evaluates the accuracy of the diagnosis by measuring various performance parameters and applying different accuracy definitions. We also present the analysis on sensor placement research in context of number of observables fed to the Lydia model.

Chapter 6 summarizes the thesis, presents the main results, and suggests future work.
Model-Based Diagnosis and Lydia

2.1 Model- Based Diagnosis

Model-based diagnosis is the process of finding faults (fault locations) in a system on basis of differences in expected and actual behavior of the system. Integrating the expected behavior of all the components in the system can derive the expected behavior of the system. Each component in the system can be modeled, as a result of which the model of the whole system evolves.

Modeling a behavior of system has many virtues. First and foremost advantage of modeling is that, the models are close to the domain of the system. The terms used in the models and the results of model outputs are directly linked to the domain. Secondly, the explanations of the models can be generated at user level. But the most significant advantage of using models is that, the results can be produced even with inexact or abstract models of the system.

2.1.1 Model-Based Diagnosis Theory

DeKleer and Williams described GDE (General Diagnostic Engine) which is able to diagnose multiple simultaneous faults in a system [6]. In a GDE model a system is partitioned in independent components which are linked together. Each component is described by a set of equations which specify the behavior of the component when it is not faulty. Nothing is assumed about a faulty component and each component has an undefined behavior. The model of the system must be free of errors, and the measurements of the system must be perfect. When an observation is recorded, GDE uses the component models to deduce the other values in the system. While doing so, it records the assumptions it makes during this inference process. Specifically these assumptions include the health state of various components. If some observables contradict the prediction, then we know that the assumption upon which the prediction was built was not true. The incorrect assumptions include a subset of components being healthy. Such a statement about the (fault) state of a set of components which explains that the observations have proved to be not true, is called a conflict. The goal of the GDE is to find a diagnosis, that can explain all the observations. A statement about the fault state which does not contain any conflict is a diagnosis. A further restriction on a diagnosis is that the diagnosis must be minimal, with the least number of faulty components.

Kleer and Kurian [5] explain the following important properties of model-based diagnosis.

- A system model is provided in terms of components and their interconnections.
- The component models describe how each component behaves.
A domain-independent reasoning engine calculates the diagnoses from the model.
The system may have multiple dependent or independent faults.
MBD does not do any pre-computation.

2.1.2 Model-Based Diagnosis Approach

Figure 2.1 shows the general diagnosis approach.

\[
\begin{align*}
    \mathbf{y} &= f(\mathbf{x}, \mathbf{h}) \\
    \mathbf{h}' &= g(\mathbf{x}, \mathbf{y})
\end{align*}
\]

Thus, the output vector is a function (f) of the input vector and the health vector.
The health of the system is then inferred from the input vector (\( \mathbf{x} \)) and the output vector (\( \mathbf{y} \)).

\[
\mathbf{h}' = g(\mathbf{x}, \mathbf{y})
\]

Thus, the diagnosis vector (\( \mathbf{h}' \)) is a function (g) of input and output vectors.
We can relate the above two functions in the following way.

\[
g = f^{-1}
\]
In model-based diagnosis approach, the diagnosis function \( g \) is not constructed directly, but from the simulation function \( f \). The diagnosis is derived by comparing actual observations with the output of simulation function \( f(z, h) \). If the observations are consistent with the predicted observations, we say that the health vector \( h \) reflects the true health of the system [4].

## 2.2 Lydia

Lydia (Language for sYstems DIAgnosis) is a model-based systems language, primarily aimed at fault diagnosis and simulation in systems [3]. It aims at the development of a systems modeling language and compiler framework that enables simulation and diagnosis of model-based systems.

### 2.2.1 Lydia Syntax

Lydia is a language in which one can specify the behavior of a system. This specification can be used for tasks such as diagnosis and simulation [10]. The statements in Lydia are declarative. All statements in a Lydia model are always true.

Let us consider the following Lydia model of an AND gate.

```lydia
system AND(
    bool in1 ,
    bool in2,
    bool out,
    bool h_AND )
{
    probability (h_AND = true) = 0.99 ;
    h_AND => (out = in1 and in2) ;
}
```

The AND system is declared along with its inputs, outputs and a health value of the AND system, as boolean values. The system is said to be healthy if the output variable \( out \) is the logical AND of both inputs \( in1 \) and \( in2 \).

An important property of the Lydia language is its ability to handle system instantiations. Thus, the components of a system can be specified in such a way that they can be reused in different places in another system.

The following Lydia model describes how the AND component can be reused to build a polycell system as shown in figure 2.2 consisting of three AND gates and two OR gates.
system AND ( 
    bool IN1 ,
    bool IN2 ,
    bool OUT ,
    bool h_AND
)
{
    probability (h_AND = true) = 0.99;
    h_AND => (OUT = IN1 and IN2) ;
}

system OR ( 
    bool IN1 ,
    bool IN2 ,
    bool OUT ,
    bool h_OR
)
{
    probability (h_OR = true) = 0.99;
    h_OR => (OUT = IN1 or IN2) ;
}
system main(
    bool OR_5_OUT,
    bool OR_4_OUT,
    bool AND_1_OUT,
    bool AND_1_IN2,
    bool AND_1_IN1,
    bool AND_2_OUT,
    bool AND_2_IN2,
    bool AND_2_IN1,
    bool AND_3_OUT,
    bool AND_3_IN2,
    bool AND_3_IN1,
    bool h_OR_5 ,
    bool h_OR_4 ,
    bool h_AND_1 ,
    bool h_AND_2 ,
    bool h_AND_3
)
{
    system AND AND_1 ;
    AND_1 ( AND_1_IN1 , AND_1_IN2 , AND_1_OUT , h_AND_1 ) ;

    system AND AND_2 ;
    AND_2 ( AND_2_IN1 , AND_2_IN2 , AND_2_OUT , h_AND_2 ) ;

    system AND AND_3 ;
    AND_3 ( AND_3_IN1 , AND_3_IN2 , AND_3_OUT , h_AND_3 ) ;

    system OR OR_5 ;
    OR_5 ( AND_1_OUT ,AND_2_OUT , OR_5_OUT , h_OR_5 ) ;

    system OR OR_4 ;
    OR_4 ( AND_2_OUT , AND_3_OUT , OR_4_OUT , h_OR_4 ) ;

}

Thus the behavior of the polycell is included in the model and the connections between the gates can be seen in the system instantiations. Lydia provides two broad features viz. simulation and diagnosis which we describe in the next section.
2.2.2 Lydia Simulator

One of the main difficulties in real world diagnosis is in deciding the test points or sensor points, which act as observation sources. In complex real world systems, it is difficult to obtain the relevant observations from the sensors because of the complexity of the system itself. In such cases, simulation is required to understand the behavior of the system, as shown in figure 2.3.

In the simulation mode (lsim), the Lydia model of the system can be compiled to a simulator in Lydia. The inputs and the health parameters can be fed to the simulator, which are used by the simulator to give corresponding outputs for the simulated system. Thus, the simulator basically solves the equations for outputs.

2.2.3 Lydia Diagnosis Engine

In the diagnosis mode (cdas), the Lydia model is diagnosed for the existing faults in the system and the fault causes are extracted in terms of the health parameters [12]. The probabilities of the health parameters (being true) are included in the model itself.

Previously diagnosis was implemented by "scotty", a program for diagnosis, which used a SAT solver to generate diagnosis [14]. It created all the possible diagnoses. This exhaustive approach was inefficient because generally, only the most probable diagnoses are significant. Also it was only usable for a maximum of six components, which is insufficient for any practical use. Hence a new algorithm for diagnosis viz. CDA* (Conflict Directed A*) [14] was implemented which generates solutions in best first order. It is applicable for more than 1000 components and minimizes the computational effort by using conflicts to jump over inconsistent solutions.

CDA* uses best first search for the diagnosis of the faults. Thus searching is an integral part of the diagnosis and it is important to keep the computational
2.2. LYDIA

complexity of the search algorithm to be low. This can be achieved by incorporating as many constraints as possible. But this solution relates to the improvement of the algorithm rather than the model. Feeding as many observations as possible to the diagnostic algorithm will be an option on model level to achieve an accurate diagnosis.

The diagnosis for the polycell model described in section 2.2.1 is shown below.

The variables AND\_2\_OUT, AND\_3\_OUT and OR\_4\_OUT are set 1, 0 and 1 respectively using set command. It is obvious that the OR\_4 gate is faulty. Thus, we inject the fault and verify it by running the cdas command. The most probable diagnosis suggested by the diagnosis engine pinpoints at the same fault, indicating that OR\_4 gate is faulty, as shown below. Thus, in real time scenario, we would get the diagnosis based on the system behavior, when the diagnosis engine is fed with the inputs and outputs along with the model of the system.

```
$ cdas and.cnf

cdas> set AND\_2\_OUT 1
@ start output <name of command should go here>
@ stop output <name of command should go here>

cdas> set AND\_3\_OUT 0
@ start output <name of command should go here>
@ stop output <name of command should go here>

cdas> set OR\_4\_OUT 0
@ start output <name of command should go here>
@ stop output <name of command should go here>

cdas> fm
@ start output <name of command should go here>
(0.00960596) h\_AND\_1=true h\_AND\_2=true h\_AND\_3=true h\_OR\_5=true h\_OR\_4=false
(9.70299e-05) h\_AND\_1=false h\_AND\_2=true h\_AND\_3=true h\_OR\_5=true h\_OR\_4=false
(9.70299e-05) h\_AND\_1=true h\_AND\_2=false h\_AND\_3=true h\_OR\_5=true h\_OR\_4=false
(9.70299e-05) h\_AND\_1=true h\_AND\_2=true h\_AND\_3=false h\_OR\_5=true h\_OR\_4=false
(9.801e-07) h\_AND\_1=true h\_AND\_2=false h\_AND\_3=true h\_OR\_5=false h\_OR\_4=false
(9.801e-07) h\_AND\_1=false h\_AND\_2=true h\_AND\_3=true h\_OR\_5=false h\_OR\_4=false
(9.801e-07) h\_AND\_1=true h\_AND\_2=false h\_AND\_3=false h\_OR\_5=true h\_OR\_4=false
(9.801e-07) h\_AND\_1=false h\_AND\_2=false h\_AND\_3=false h\_OR\_5=false h\_OR\_4=false
(9.9e-09) h\_AND\_1=false h\_AND\_2=false h\_AND\_3=false h\_OR\_5=false h\_OR\_4=false
(9.9e-09) h\_AND\_1=false h\_AND\_2=true h\_AND\_3=false h\_OR\_5=false h\_OR\_4=false
(9.9e-09) h\_AND\_1=false h\_AND\_2=true h\_AND\_3=true h\_OR\_5=false h\_OR\_4=false
(1e-10) h\_AND\_1=false h\_AND\_2=false h\_AND\_3=false h\_OR\_5=false h\_OR\_4=false
@ stop output <name of command should go here>
```
The above result gives a list of most probable diagnoses in the decreasing order of the probability. This probability depends on the behavior of the model and the a-priori probabilities of the health variables defined in the system model.

In the next chapter, we introduce the ASML lithography machine and describe the details of E-pins case study that will be solved using model-based diagnosis concept and the Lydia model.
3.1 Introduction to ASML lithography Machine

ASML lithography systems use a photographic process to image nanometric circuit patterns onto a silicon wafer [1]. An exact dose of light energy should be provided in order to properly expose the wafer. A wafer stage moves the wafer in scanning motion underneath the lens that focuses the light. We consider a case study for diagnosis that is related to a sub-part of the wafer stage viz. E-pins. We focus on a specific wafer positioning error (E-pin Error) in the case study that will be explained in section 3.7.

Following are various sub-parts of the machine that are relevant to the E-pin case study [9].

- Wafer handler
  The wafer handler transports the wafer to the track on the machine, places the wafer on the wafer stage for measurement and exposure and then removes the wafer from the machine.

- Wafer stage
  There are two wafer stages in the machine. One wafer stage is used for measuring the wafer while the other wafer stage is used for exposing the wafer. On the exposure side, the wafer stage is responsible for accurate positioning of the wafer beneath the lens for exposure and exchanging the wafer with the wafer handler. Additionally, it also provides an interface for several sensors like various temperature sensors and position sensors.

- Chuck
  The chuck provides a base on which wafer is placed for exposure. It is located on the wafer stage.

- E-pin
  Three E-pins extend through the wafer table and clamp the wafer while exchanging it with the wafer handler. Vacuum on the tip of the E-pin tip helps in facilitating the clamping process. E-pins are slide up to the wafer level for clamping. E-pin up and E-pin down sensors keep the track of the position of the E-pin.
3.2 Wafer Exchange Process

In order to speed up the wafer exposure process, the current lithographic machine uses two wafer stages. Thus, the measuring and the exposing of the wafer can be done in parallel. Between measurement and exposure, the whole chuck is swapped.

In wafer load/unload procedure, the chuck is moved to the predefined position. Vacuum is established on the E-pins and a lock is activated so that chuck gets locked in its current position. The wafer is lifted to the maximal height by moving the E-pins fully up and removed from the chuck. The E-pins’ vacuum is released and the E-pins are lowered fully down.

3.3 Hardware Description

The infrastructure that is implemented for machine safety involves a number of boards as shown in figure 3.3 [8].

![Figure 3.1: Hardware Infrastructure](image)

These boards include,

* Rail Dedicated Logic Board (RDLB)
  The RDLB executes the machine damage control task for wafer stage. It checks the validity of software commands to operate the stage, based on
real time obtained sensor information. As such, the RDLB safeguards the chuck swap.
To obtain a high level in machine damage control performance, the RDLB is specially designed to check upon its own integrity and to force the stage to a safe state when finding an DLB-internal failure.

* Local Sensor Board (LSB)
The LSB hosts the E-pin circuitry that includes the E-pin position sensors, exchange sensor and E-pin actuator. E-pin position sensors include E-pin Up and E-pin Down sensors. Exchange sensor detects whether the chuck is at exchange position or not. E-pin motor is controlled by the E-pin enable signal. E-pin enable signal is TRUE if the E-pins are clamped to the wafer using vacuum.

* Sensor Board (SB)
The SB keeps track of states of various sensors like temperature sensors, position sensors and pneumatic levels on the wafer stage.

* Power Amplifier Controller (PAC)
The PAC acts as an interface between the motion controller hardware and wafer stage power amplifiers. The motion controller comprises of RDLB, LSB and SB. The PAC sends stop signals to the wafer stage power amplifiers in case of an emergency to halt the working of the wafer stage.

* Chuck Dedicated Logic Board (CDLB)
Whenever an artifactous situation is detected by any of the boards an emergency ripples through the wafer stage. The CDLB plays a key role in the distribution of emergencies. In case that an emergency is detected by any of the RDLB, LSB or SB, the PAC is notified by the CDLB and an emergency stop is initiated.

3.4 E-PIN Safety Logic
The E-pin module is in the center of the chuck and it accepts wafers from the wafer handler and lifts up the wafers to make the wafer accessible to the wafer handler. The pins extend through the wafer table without making contact with it. Position detecting sensors ensure that the E-pins are fully up or down. The E-pins are raised further once the wafer handler has entered and is above the wafer table. Vacuum is applied through each hollow pin to secure the wafer to the pins as the pins lift the wafer from the wafer handler during measurement or exposure. As the pins lower to their down position below the wafer table, this vacuum is switched off and vacuum is applied through the wafer table to ensure that the wafer does not move on the wafer table during exposure.
During the above process, E-pins should always be down if the chuck is not at the exchange position. If not, an error is issued by the LSB to the PAC and an emergency stop is activated. This error is known as the E-pin error.
Figure 3.4 shows the E-pin hardware schematic. The LSB hosts the E-pin circuitry. The E-pin circuitry includes the Exchange sensor, the E-pin up sensor and the E-pin down sensor. The corresponding signals from these sensors are sent to RDLB. RDLB issues the E-pin enable signal to LSB. The E-pin error can be represented in terms of these signals as follows.

\[
\text{EPIN ERROR} = \text{NOT}(\text{E-pin Down}) \text{ AND NOT}(\text{Chuck at Exchange})
\]

### 3.5 Extracting Data from Board Dumps

Board dumps from LSB can be used to diagnose the E-pin error. Actual data from the machine can be extracted and fed as input to the model. We show a case where the E-pin error had occurred, in section 3.7.3. The states of various sensors of the LSB (Local Sensor Board) are dumped into the registers of LSB. These registers are described by the Hardware-Software Interface (HSI). The values of these registers are available in the board dumps of the LSB, which are converted to the Lydia data by simple python code. Table 3.1 shows the relevant sensors to the E-Pin case, the register name in which they can be located and the bit in the register that corresponds to the sensor value [7].

The above errors are said to have occurred if the corresponding register bits are TRUE.
3.6 Why Diagnose with Lydia Model?

Currently, ASML uses lookup tables in form of Excel sheets, to match the fault symptoms with the actual faults as shown in figure 3.5. This lookup process can be time consuming and inaccurate. Furthermore, there is a possibility of the manual errors while inferring the faults. The effects of the errors in the fault inference could lead to unnecessary overheads in terms of cost and time. Thus, there is scope for improvement in speeding up the fault diagnosis procedure and increasing the accuracy of the fault diagnosis.

Modelling the errors like E-pin error in Lydia and diagnosing them using the diagnosis engine eases the fault finding procedure. The correct behavior of the system is inserted in the model. Thus, there is no need to keep track of every possible error and its corresponding symptoms. The faults are inferred by the diagnosis engine when it is fed with the Lydia model and the input sensor values. The time required for fault diagnosis is equal to the time taken by the diagnosis engine to infer the fault, which is in terms of several seconds. Furthermore, since the fault location is inferred by the diagnosis engine, there is no possibility of manual errors.

Hence, we model the E-pin behavior in a Lydia framework that is extracted from the Electrical Layouts which represents the hardware description of various electrical boards described in section 3.3. The detailed information about extraction process from the Electrical Layouts is described in the next chapter.

3.7 Why (Semi-)Automate Lydia Model Generation?

The accuracy of the diagnosis depends on the accuracy of the system behavior that is inserted in the Lydia model. Usually the system behavior is manually inferred and inserted into the Lydia model. The accuracy of the human inference of the system behavior depends on the expert. (Semi-)Automating the Lydia model generation can remove the errors related to inference about the structure of the system.

We present the E-pin Lydia model inferred by expert and the diagnosis of the E-pin error using it. In chapter 4, we will present the diagnosis using the
CHAPTER 3. ASML LITHOGRAPHY MACHINE - E-PINS CASE STUDY

Figure 3.3: Current ASML Diagnosis Method Vs. Diagnosis with Lydia

E-pin Lydia model which is (semi-)automatically generated and show that the diagnosis given by both the models is equivalent, thus stressing the usefulness of (semi-)automatic model extraction.

3.7.1 E-pin Lydia Model Inferred by Human Expert

system ws_lsb (  
  bool h_sensor_up, h_sensor_down,  
  bool h_exchange_sensor,  
  bool h_safety_relay,  
  bool h_lsb, h_rdlb,  
  bool epin_up, epin_down, epin_enable,  
  bool chuck_at_wafer_exchange,  
  bool epin_error )  
{  
  // e-pin position sensor
3.7. WHY SEMI-AUTOMATE LYDIA MODEL GENERATION?

```
h_sensor_up => (epin_up = real_epin_up)
h_sensor_down => (epin_down = real_epin_down)

// chuck at exchange sensor
h_exchange_sensor =>
    (chuck_at_wafer_exchange = real_chuck_at_wafer_exchange)

// ASSUMPTION: if the safety relay fails, it opens and the epin
// goes down because of gravity the epin_enable is independent
// it comes from the rail DLBL
if (h_safety_relay) (not(epin_enable) => real_epin_down)
else (real_epin_down=true)

h_lsb => (epin_error=(not(epin_down) and not(chuck_at_wafer_exchange)))

h_rdlb => (epin_enable => real_chuck_at_wafer_exchange)

// the real epin position
// enum of up middle down
(real_epin_up and not(real_epin_middle) and not(real_epin_down)) or
(not(real_epin_up) and real_epin_middle and not(real_epin_down)) or
(not(real_epin_up) and not(real_epin_middle) and real_epin_down)

// h variables
probability (h_sensor_up=true)=0.99
probability (h_sensor_down=true)=0.99
probability (h_exchange_sensor=true)=0.99
probability (h_safety_relay=true)=0.99
probability (h_lsb=true)=0.99
probability (h_rdlb=true)=0.99
```
3.7.2 Model Details

Following details are inserted in the above model (wafer stage system).

* If the safety relay (switch controlling the E-pins) is open, the E-pins are down because of gravity.
* If the E-pins are enabled, the safety relay is closed.
* If the RDLB is healthy, it only enables the E-pins if the chuck is locked, i.e. the chuck is physically at the exchange position.
* E-pin error occurs when the E-pin is not down and the chuck is not at the exchange position.

Since, the ASML machine has two wafer stages, the wafer stage system is instantiated twice.

3.7.3 Diagnosis

The model is diagnosed for two states, derived from the Wafer Stage dump files\(^1\).

\[\begin{align*}
\text{h1}_\text{sensor}_\text{up} & [0.99] \\
\text{h1}_\text{sensor}_\text{down} & [0.99] \\
\text{h1}_\text{exchange}_\text{sensor} & [0.99] \\
\text{h1}_\text{safety}_\text{relay} & [0.99] \\
\text{h1}_\text{lsb} & [0.99] \\
\text{h}_\text{rdlb} & [0.99] \\
\text{h2}_\text{sensor}_\text{up} & [0.99] \\
\text{h2}_\text{sensor}_\text{down} & [0.99] \\
\text{h2}_\text{exchange}_\text{sensor} & [0.99] \\
\text{h2}_\text{safety}_\text{relay} & [0.99]
\end{align*}\]

\(^1\)Note that the diagnosis represented is a result of older version of cdas, and hence differs from the current diagnosis representation. This is because the E-pin model inferred by the expert was compatible to older version of cdas and since then cdas has been upgraded to a newer version.
3.7. WHY (SEMI-)AUTOMATE LYDIA MODEL GENERATION?

h2_lsb [0.99]
# n -i time epin_error1 epin_up1 epin_down1 chuck_at_wafer_exchange1 epin_enable1 epin_error2 epin_up2 epin_down2 chuck_at_wafer_exchange2 epin_enable2
# d 0.0 1 0 0 0 0 0 0 1 0 0
d 0.000000 0.00904382 1 0 1 1 1 1 1 1 1 1 1
d 0.000000 9.13517e-05 1 0 0 1 1 1 1 1 1 1 1
d 0.000000 9.13517e-05 1 0 1 0 1 1 1 1 1 1 1
d 0.000000 9.13517e-05 1 0 1 1 0 1 1 1 1 1 1

The above diagnosis shows that if the epin_error1 (epin_error for wafer stage 1) is present, then the diagnosis suggests that the Epin down sensor is faulty.
LYDIA Model Generation from Electrical Layouts

4

4.1 Introduction

Electrical layouts (E-Layouts) are the electrical drawings that represent a complete functional overview of all the electrical functions of a system. E-Layouts ensure that the electrical infrastructure of the subsystem and its modules, is designed, prototyped and qualified in a controlled way.

Various tools like AutoSketch, AutoCAD Electrical, Viewlogic etc support the development of E-Layouts. ASML uses Viewdraw (V5.4.3) which is a part of Viewlogic software package [2]. ViewDraw is a graphical tool for design capture and analysis of E-Layouts. It can be used for digital, analog, or mixed-signal circuitry entry at the system, printed circuit board, or integrated circuit levels. ViewDraw is also tightly linked to synthesis, simulation, timing analysis, and layout thus supporting the dynamic analysis, verification, and debugging throughout the design process.

All designs in Viewdraw can be represented in two ways viz. graphical and textual. In this chapter, we explain how the information is extracted from the textual view of the E-layouts and converted into the Lydia models.

4.2 E-layout Structure

* Symbol
  - It is a graphical representation of a functional object including connections and its parameters.

* Component
  - Component is an instance of a symbol used in a schematic.
  - Each instance is a separate image of the symbol.

* Block
  - It is the sheet area which the symbol occupies. Block is also known as the Bounding Box.

* Net
  - Net is a graphical representation of the signal interconnection of a set of component pins.
CHAPTER 4. LYDIA MODEL GENERATION FROM ELECTRICAL LAYOUTS

Figure 4.1: E-layout Anatomy

- A Net can have multiple branches.
- A branch connected to only one pin is called a dangling branch.

* Bus
  - Bus is a graphical representation of a set of nets.
  - It is analogous to a cable.

* Pin
  - The symbol object which serves as the interconnection between a Net or Bus and the symbol is known as a pin.
  - All symbol pins must have a user-specified label.

* Attribute
  - A text string which contains parametric information about the symbol or one of its objects is known as attribute.

Figure 4.1 shows the E-layout anatomy for polycell. The polycell E-layout is enclosed in a block which comprises of 5 components (3 AND gates and 2 OR gates). These components are connected by lines called nets. Every component and a net has a unique internal Viewdraw id. The nets are connected to the components by pins. Each pin also has a unique internal id which should be extracted from the symbol section.
4.3 E-Layout File Format

ViewDraw stores information about schematics and symbols as ASCII text files. This is the textual view of the E-layouts. The information is stored in a line oriented fashion, with each line beginning with a special character (normally an alphabet) which declares an item. It is followed by the attributes of the item (like co-ordinates, item id etc.) in the remainder of the line. These items include nets/bus/symbol declarations, their labels, connection information etc.

These ASCII text files comprise of two sections. One section describes the schematic details while the other one describes the symbol details.

We develop a detailed grammar for this view described in Appendix A. The grammar would explain the meaning of each ASCII line. The textual view for the overall polycell E-layout anatomy in figure 4.1 is as follows.

```
V 50
K 164960849300 polycell
Y 0
D 0 0 2338 1653
Z 7
i 16
b 480 800 950 1250
N 14
J 780 880 3
J 780 950 3
J 640 880 2
J 800 950 2
S 3 1
S 1 2
S 2 4
N 12
J 730 990 3
J 730 1040 5
J 790 1040 3
J 790 1110 3
J 650 1040 2
J 810 1110 2
J 800 990 2
S 1 2
S 1 7
S 5 2
S 2 3
S 3 4
S 4 6
N 10
```
The following information should be extracted in order to build a Lydia framework from the Viewdraw textual format. The exact relation between an item in the E-layout (component, net, bus etc.) and its corresponding ASCII line can be found in Appendix A.

* The component names along with their labels (if any) and the respective component id’s which represent the instantiation of the component. This information can be extracted from the symbol section of the textual format file.

* The net/bus labels and their corresponding id’s

* The connection points where a net/bus is connected to the pin of the component. This is identified by comparing the co-ordinates of each net/bus end with the co-ordinates of each pin of every component. To detect multiple net connections, co-ordinates of each net/bus end should be tracked for a connection with a net instead of a pin.

The other information in E-layouts such as bounding boxes, joints, color of
4.4 Lydia Model Generation Process

4.4.1 Introduction

![Figure 4.2: Lydia Model Extraction from E-layout Design](image)

Figure 4.2 shows the generic block diagram for Lydia model extraction process. The schematic ASCII file and the symbol files of the E-Layout are fed to the translator which translates the information contained in them, into a Lydia model.

4.4.2 Translator Tool Details

A two pass translator is built to carry out the conversion of ASCII Viewdraw file to the Lydia model.

4.4.2.1 PASS 1

An intermediate file is built in this pass, which contains the information extracted from the ASCII file in the following form.

Table 4.1 shows the intermediate file for the Polycell E-Layout design shown in figure 4.1.
Table 4.1: Intermediate Format after Pass 1

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Id</th>
<th>Pin Label</th>
<th>Connection Net Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>5</td>
<td>IN2</td>
<td>14</td>
</tr>
<tr>
<td>OR</td>
<td>5</td>
<td>IN1</td>
<td>12</td>
</tr>
<tr>
<td>OR</td>
<td>4</td>
<td>IN2</td>
<td>12</td>
</tr>
<tr>
<td>OR</td>
<td>4</td>
<td>IN1</td>
<td>10</td>
</tr>
<tr>
<td>AND</td>
<td>3</td>
<td>OUT</td>
<td>10</td>
</tr>
<tr>
<td>AND</td>
<td>2</td>
<td>OUT</td>
<td>12</td>
</tr>
<tr>
<td>AND</td>
<td>1</td>
<td>OUT</td>
<td>14</td>
</tr>
</tbody>
</table>

### 4.4.2.2 PSEUDO CODE for PASS 1

```python
for every_asci_line in asci_file {
    if component_found in asci_line {
        component_name = extract_name(asci_line)
        component_id = extract_id(asci_line)
        pins[] = extract_pin_info_from_symbol_section(component_name)
        /* pin_info = pin_id + pin_label */
        for pin_label in pins[]
            { 
                nets[] = get_connection_info(pin_label)
                /* connection_info = net_label */
                for net in nets[]
                    { 
                        net_id = get_net_id(net)
                        intermediate_file_format.append (component_name, component_id, pin_label, net_id)
                    }
            }
    }
}
```

### 4.4.2.3 PASS 2

The intermediate file is processed to find out the "Component Pins" connected to the same connection net id. Further the connections are represented in a Lydia model.

### 4.4.2.4 PSEUDO CODE for PASS 2

```python
for each unique component_name in intermediate_file_format
```
4.4. LYDIA MODEL GENERATION PROCESS

```c
{
    pins[] = get_pin_info_from_library(component_name)
    /* pin_info = pin_id + pin_label */
    build_lydia_system(component_name, pins[])
}

group_same_component_id_pins ()

for each group
{
    find_no_of_nodes()
    build_lydia_system_with_behavior_for_nets()
    instantiate_each_connection()
}
```

The behavior of the net system is automatically inserted, while the behavior of the other components should be inserted manually. The behavior of the net system with $n$ different nets can be automated as follows.

```c
system net_n_nodes {
    for i=1 to n
    {
        bool node_i
    }
    bool h_net_n_nodes )
{
    probability (h_net_n_nodes = true) = 0.99 ;
    bool connection ;
    for i=1 to n
    {
        h_net_n_nodes => (connection = node(i)) ;
    }
}
```

4.4.3 Introducing Hierarchy into Lydia Model

Given a device to be diagnosed, a key problem is to find a decomposition that will be useful for diagnostic problem solving. The behavior of models and the observable conditions at each level of abstraction should be available. Moreover, this information must be in a format that is compatible to the diagnosis engine so that the faults can be efficiently isolated.

E-Layouts can be hierarchically modeled in the following way, which helps the diagnosis engine in reducing the search space.

The components in the E-Layout are decomposed into a hierarchy of sub-components. Hierarchy of the E-Layout design can be exploited by first identifying the high-level components and relation between those components.
Then the behavior of each component can be inserted manually or automatically (if the component description input is compatible to the translator tool) for further focused diagnosis. Each component is considered as a separate system and is instantiated at every occurrence of a connection. This abstraction can be extended by classifying the connections between the components into separate systems. The classification is based on the number of nodes for each connection.

Additionally, the function of a component can be described with different levels of precision which can be achieved in the connection systems by abstracting multiple nets connecting the same components into a bus.

### 4.4.4 Conventions Used in Lydia Model Creation

* The system names for components are modeled from the component names used in Viewdraw textual files. These are typically names of symbol files.
* The health variable of the component system is modeled as a variable $h_{\text{Component\_System\_Name}}$.
* The component system is instantiated by appending $\_\text{Component\_Id}$ to the original component system name.
* The systems representing the nets with $n$ nodes are modeled as $\text{net}_n$.
* The health variable of the net system is modeled as $h_{\text{Net\_System\_Name}}$.
* The net system is instantiated by appending $\_\text{Net\_Id}$ to the original net system name.

The LYDIA model extracted from the Polycell E-Layout is shown below.

```plaintext
system OR (bool IN1 ,
        bool IN2 ,
        bool OUT ,
        bool h_OR
    )
{
    probability (h_OR = true) = 0.99;
}

system AND (bool IN1 ,
        bool IN2 ,
        bool OUT ,
        bool h_AND
    )
{
    probability (h_AND = true) = 0.99;
}
```
4.4. LYDIA MODEL GENERATION PROCESS

```c
system net_2_nodes {
    bool node_1,
    bool node_2,
    bool h_net_2_nodes
}
{
    probability (h_net_2_nodes = true) = 0.99 ;

    bool connection ;
    h_net_2_nodes => (connection = node_1 ) ;
    h_net_2_nodes => (connection = node_2 ) ;
}

system net_3_nodes {
    bool node_1,
    bool node_2,
    bool node_3,
    bool h_net_3_nodes
}
{
    probability (h_net_3_nodes = true) = 0.99 ;

    bool connection ;
    h_net_3_nodes => (connection = node_1 ) ;
    h_net_3_nodes => (connection = node_2 ) ;
    h_net_3_nodes => (connection = node_3 ) ;
}

system main{
    bool OR_5_OUT,
    bool OR_5_IN2,
    bool OR_5_IN1,
    bool OR_4_OUT,
    bool OR_4_IN2,
    bool OR_4_IN1,
    bool AND_1_OUT,
    bool AND_1_IN2,
    bool AND_1_IN1,
    bool AND_2_OUT,
    bool AND_2_IN2,
    bool AND_2_IN1,
    bool AND_3_OUT,
}
bool AND_3_IN2,
bool AND_3_IN1,
bool h_OR_5,
bool h_OR_4,
bool h_AND_1,
bool h_AND_2,
bool h_AND_3,
bool h_net_2_nodes_10,
bool h_net_3_nodes_12,
bool h_net_2_nodes_14)
{
    system OR OR_5;
    OR_5 (OR_5_IN1, OR_5_IN2, OR_5_OUT, h_OR_5);

    system OR OR_4;
    OR_4 (OR_4_IN1, OR_4_IN2, OR_4_OUT, h_OR_4);

    system AND AND_1;
    AND_1 (AND_1_IN1, AND_1_IN2, AND_1_OUT, h_AND_1);

    system AND AND_2;
    AND_2 (AND_2_IN1, AND_2_IN2, AND_2_OUT, h_AND_2);

    system AND AND_3;
    AND_3 (AND_3_IN1, AND_3_IN2, AND_3_OUT, h_AND_3);

    system net_2_nodes net_2_nodes_10;
    net_2_nodes_10 (OR_4_IN2, AND_3_OUT, h_net_2_nodes_10);

    system net_3_nodes net_3_nodes_12;
    net_3_nodes_12 (OR_4_IN1, OR_5_IN2, AND_2_OUT,
                    h_net_3_nodes_12);

    system net_2_nodes net_2_nodes_12;
    net_2_nodes_12 (OR_5_IN1, AND_1_OUT, h_net_2_nodes_12);

    system net_2_nodes net_2_nodes_14;
    net_2_nodes_14 (OR_5_IN1, AND_1_OUT, h_net_2_nodes_14);
}

The behavior of the low level components like OR and AND gates has to be inserted manually in order to make this model equivalent to the polycell model described in section 2.2.1. The reason for necessity of manual insertion of behavior for the low level components like OR and AND gates is that the E-layouts do not show the internal structure of these low level components.
4.5 ASML Case Study - WS Model

4.5.1 Extraction of E-pin Model

Figure 4.3: E-pin E-layout Representation in Viewdraw

Figure 4.3 shows the E-pin E-layout graphical representation as seen in Viewdraw. Following components are extracted from the textual format of figure 5.3.

- Local Sensor Board
- E-pin Motor
- E-pin up sensor
- E-pin down sensor
- Chuck exchange sensor

Additionally, the connection points visible in figure 4.3 are also extracted. The E-pin model extracted from the E-layout is described in Appendix B. In order to execute the fault diagnosis, the structural framework is not enough and has to be supplemented with the behavior of the sensors. Only the behavior of the E-pin up sensor, E-pin down sensor, Exchange sensor and the E-pin motor is included in the model and the behavior of the Local sensor board components is not inserted in the model since it is not related to the actual E-pin case study.
4.5.2 Diagnosis of E-pin Model

The aim of (semi-)automatically extracting the model is to increase the structural accuracy of the model. In this section we show that the diagnosis result for such a (semi-)automatically extracted E-pin Lydia model is equivalent to the diagnosis result for the model that is inferred by the human expert that is described in section 3.7.3. Thus, we demonstrate the usefulness of (semi-)automatic extraction.

\[
\begin{align*}
\text{(8.51458e-05)} & \quad h_{\text{DWSENS_VSO_IND_2W_2536}} = \text{true} \\
& h_{\text{DWSENS_VSO_IND_2W_2535}} = \text{true} \\
& h_{\text{DWSENS_VSO_IND_2W_1561}} = \text{false} \\
& h_{\text{ACT_MOTOR_DC_COIL_3210}} = \text{true} \\
& h_{\text{SENS_POS_LVDT_PRIM_SEC_2739}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIUNBAL_17_24_2669}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIBAL_1_8_2667}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIUNBAL_9_16_2671}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIBAL_1_8_2667}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIUNBAL_9_16_2671}} = \text{true} \\
& h_{\text{net_2_1459}} = \text{true} \\
& h_{\text{net_2_1468}} = \text{true} \\
& h_{\text{net_2_1484}} = \text{true} \\
& h_{\text{real_behavior}} = \text{false} \\
& h_{\text{real_position}} = \text{false} \\
\end{align*}
\]

\[
\begin{align*}
\text{(8.60058e-07)} & \quad h_{\text{DWSENS_VSO_IND_2W_2536}} = \text{false} \\
& h_{\text{DWSENS_VSO_IND_2W_2535}} = \text{true} \\
& h_{\text{DWSENS_VSO_IND_2W_1561}} = \text{false} \\
& h_{\text{ACT_MOTOR_DC_COIL_3210}} = \text{true} \\
& h_{\text{SENS_POS_LVDT_PRIM_SEC_2739}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIUNBAL_17_24_2669}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIBAL_1_8_2667}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIUNBAL_9_16_2671}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIBAL_1_8_2667}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIUNBAL_9_16_2671}} = \text{true} \\
& h_{\text{net_2_1459}} = \text{true} \\
& h_{\text{net_2_1468}} = \text{true} \\
& h_{\text{net_2_1484}} = \text{true} \\
& h_{\text{real_behavior}} = \text{false} \\
& h_{\text{real_position}} = \text{false} \\
\end{align*}
\]

\[
\begin{align*}
\text{(8.60058e-07)} & \quad h_{\text{DWSENS_VSO_IND_2W_2536}} = \text{false} \\
& h_{\text{DWSENS_VSO_IND_2W_2535}} = \text{true} \\
& h_{\text{DWSENS_VSO_IND_2W_1561}} = \text{false} \\
& h_{\text{ACT_MOTOR_DC_COIL_3210}} = \text{true} \\
& h_{\text{SENS_POS_LVDT_PRIM_SEC_2739}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIUNBAL_17_24_2669}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIBAL_1_8_2667}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIUNBAL_9_16_2671}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIBAL_1_8_2667}} = \text{true} \\
& h_{\text{PCA_MIX_LSB2_AIUNBAL_9_16_2671}} = \text{true} \\
& h_{\text{net_2_1459}} = \text{true} \\
& h_{\text{net_2_1468}} = \text{true} \\
& h_{\text{net_2_1484}} = \text{true} \\
\end{align*}
\]
In the above diagnosis result, we see very complicated variable names as they are generated while information extraction from the E-layouts. To enable the reader to comprehend the diagnosis, we present the relevant variable names with their meanings. Refer to appendix B for the E-pin Lydia model.

\[ h_{\text{real\_behavior}} = true \quad h_{\text{real\_position}} = true \]

\[ (8.60058e-07) \quad h_{\text{DW\_SENS\_VS0\_IND\_2W\_2536}} = false \]
\[ h_{\text{DW\_SENS\_VS0\_IND\_2W\_2535}} = true \]
\[ h_{\text{DW\_SENS\_VS0\_IND\_2W\_1561}} = false \]
\[ h_{\text{DW\_ACT\_MOTOR\_DC\_COIL\_3210}} = true \]
\[ h_{\text{DW\_SENS\_POS\_LVDT\_PRIN\_SEC\_2739}} = true \]
\[ h_{\text{DW\_PCA\_MIX\_LSB2\_AIUNBAL\_17\_24\_2669}} = true \]
\[ h_{\text{DW\_PCA\_MIX\_LSB2\_AIRBAL\_1\_8\_2667}} = true \]
\[ h_{\text{DW\_PCA\_MIX\_LSB2\_AIUNBAL\_9\_16\_2671}} = true \]
\[ h_{\text{net\_2\_1459}} = false \quad h_{\text{net\_4\_1468}} = true \quad h_{\text{net\_2\_3689}} = true \]
\[ h_{\text{net\_2\_1315}} = true \quad h_{\text{net\_2\_1432}} = true \quad h_{\text{net\_2\_1482}} = true \]
\[ h_{\text{net\_2\_1488}} = true \quad h_{\text{net\_2\_1483}} = true \quad h_{\text{net\_2\_1484}} = true \]
\[ h_{\text{net\_2\_1451}} = false \quad h_{\text{real\_behavior}} = false \]
\[ h_{\text{real\_position}} = false \]

4.5.3 Fault Models

The real defects in the system are too numerous and often not analyzable. Hence, we need to isolate the targets of testing for faults. A fault model identifies such targets for testing the faults.

Although there are far too many different fault models, we focus on stuck-at faults and bus-faults.

* Stuck-at Fault

This fault is modeled by assigning a fixed (0 or 1) value to a signal line in the circuit. A signal line is an input or an output of a component. The most popular forms are the single stuck-at faults, i.e. two faults per net,
stuck-at-1 and stuck-at-0. The stuck-at fault is the type of a structural
fault.

* Bus Fault
  A bus fault specifies the status of each line in a bus as stuck-at-1, stuck-
  at-0 or fault free. Thus, for a n-bit bus, there are $3^n$ bus faults. A total
  bus fault assumes all the lines of the bus to be stuck at the same 1 or 0
  state.

4.5.4 Relating Fault Models to E-pin Lydia Model

If a particular sensor output is stuck at 0 or 1, it can be modeled as a stuck-
at-fault. This fault can be extended to the bus fault by modeling the nets
with multiple connection with stuck-at-faults.

Thus from the diagnosis shown in section 4.5.2, we can interpret that if
$h_{\text{Dw_SENS_VSO_IND_2W_2536}}$ is false, it can be either stuck-at 0 or the sensor
is giving false value based on the actual behavior of the sensor.

In case of nets, if the health variable representing the net is 0, then it can be
modeled as a bus fault.
In this chapter we evaluate the diagnosis accuracy based on various metrics. The terms and metrics used for diagnosis evaluation as described in the following section.

### 5.1 Terms and Metrics

- **Inputs to the Model (x)**
  These are the component input variables which should be fed to the simulator along with the health variables to find out the observables.

- **Observables (y)**
  Observables are the observations/variables in the model whose values are fed to the diagnosis engine. The values of output health parameters depend on observables.

- **Components (m)**
  Components are the sensors in the system.

- **Injected Health Vector (h = (h₁...hₗ...hₘ))**
  Health vector is a vector consisting of all the health variables which are under observation and it is the eventual output of the diagnosis along with its probability. Injected health vector is the health vector fed to the simulator (lsim).

- **Output Health Vector (H = {ₗ₁...ₖ₂...ₖₙ})**
  It is a vector of health vectors. H contains the diagnosis of faults in the model. This vector is the output of the diagnosis engine with n diagnoses.
• **Output Probability Vector** \( \hat{P} = \{\hat{p}_1 \ldots \hat{p}_i \ldots \hat{p}_n\} \)
  This vector contains the normalized diagnosis probability corresponding to each diagnosis health vector.

• **Diagnosis Entropy** \( \hat{e} \):
  Diagnosis entropy is the diagnostic measure which represents uncertainty of the diagnosis [13].

  \[
  \hat{e} = \sum_{i=1}^{n} -\hat{p}_i \log \hat{p}_i
  \]

  where \( \hat{p}_i \) is the probability at position \( i \) in the diagnosis array.

• **Relative Observability** (R)
  This indicates the average number of observations per component (health variable). It can be formulated as,

  \[
  R = \frac{|y|}{|h|}
  \]

  Thus, it gives indication of density of health components in the model.

• **Accuracy of Diagnosis and Accuracy Model**
  Accuracy indicates the closeness of the true or accepted value to the measured value.
  The diagnosis performance can be realistically checked for accuracy in real life, if there is provision to crosscheck whether the diagnosis is correct, by accessing the fault-diagnosed parts in the real system.
  For example, if the diagnosis system evaluates that the exchange sensor in the E-pin module is faulty, then there should be provision to have access to the exchange sensor and check it for performance.
  But for these experiments, the realistic access to all the system sub-parts, which are being represented by health parameters, is not feasible. Hence there is no possibility of comparison of diagnosed faults with the actual realistic sub-system. In such case the simulator plays a significant role in simulating the system and predicting the sensor values based on input values and assumed healths of the components. The diagnosis as a function of observability has to be cross checked with the simulator values in order to get the percentage accuracy of the diagnosis.

  * Definition 1
  
  Accuracy for \( \hat{h}_i \):

  \[
  a_i = \frac{1}{m} \sum_{j=1, h_{ij}=h_j}^{m} 1
  \]

  In this definition of the accuracy we give weight to the number of health variables matching for each diagnosis vector in the diagnosis
array with the injected health vector.

* Definition 2

Score for $h_j$:

$$s_j = \sum_{i=1, h_{ij} = h_j}^{n} \hat{p}_i$$

Score for column position $j$ is the sum of probabilities of matching diagnosis for position $j$ in each health vector.

Accuracy for $\hat{h}_j$:

$$Accuracy = \frac{\sum_{j=1}^{m} s_j}{m}$$

In this definition of the accuracy, we give weight to the sum of the probabilities corresponding to the matching health variable positions and calculate the mean value of score total.

5.2 Sensor Placement

5.2.1 Motivation for Experimentation

One of the main difficulties in real world diagnosis is in deciding the test points or sensor points, which act as observation sources. In complex real world systems, it is difficult to obtain the relevant observations from sensors because of the complexity of the system itself. Thus, even if the sensor is most suitable to diagnose a certain parameter, it might not be feasible to get observation from that sensor. Hence in that case, the diagnosis algorithm cannot be fed with that observation, which eventually affects the accuracy of the diagnosis.

Hence, it would be interesting to note how the accuracy of the diagnosis varies, as the number of observations from the sensor elements are varied. This motivates the investigation of the performance of diagnosis as a function of observability. The above investigation would make it possible to decide the minimum number of observations to be taken into account, given the permissible range of the accuracy. This would also help in deciding the importance of the sensor elements, which are practically not accessible. As a result, unnecessary waste of effort and cost in accessing useless sensor points can be restricted.
Also it would be interesting to note the variation in accuracy with respect to the entropy of the diagnosis. We can thus predict the uncertainty of the diagnosis given the accuracy of the diagnosis.

5.2.2 Experimental Setup

The experimental setup designed to carry out the experiments is model independent. Thus the same experiments can be carried out for any other model with little or no change in the set up.

The experimental setup aims at building up a cycle that would accept the input health vector for the simulator and provide with the accuracy of the diagnosis along with corresponding variance in accuracy, for a finite number of runs, with the number of observables increased linearly during each run. Each time the observables are fed to the diagnosis engine in random order.

Pseudo Code

The following pseudo code describes the experimentation setup and indicates how accuracy and variance in accuracy are calculated for an injected health vector.

inputs : h_vec - injected health vector

outputs : Accuracy - Accuracy array of Diagnosis
  Variance - Variance array in Accuracy due to randomization

Variable info : x_vec - inputs to simulator
  y_vec - outputs from simulator
  H_vec - diagnosis vector
  P_vec - probability vector corresponding to diagnosis

for every h_vec with zero, one or two faults
{
  x_vec = NULL
  y_vec = lsim (h_vec, x_vec)
  for (i = 1 to mod(y_vec) )
  {
    for ( j = 0 to finite_runs_number-1 )
    { /* Randomization will be achieved finite_runs_number of times */
      y_in = random(y_vec, i)
      /* Function random returns i random values from y_vec vector */
      (H_vec, P_vec) = cdas(x_vec, y_in)
      pos = find_position_of_injected_h(h_vec,H_vec)
5.2. SENSOR PLACEMENT

```cpp
prob = find_probability_of_injected_h(P_vec, pos)
sum = sum_of_all_probabilities(P_vec)
A(j) = calculate_accuracy(prob, sum)
}
accuracy(i) = avg_accuracy(A)
entropy(i) = calculate_entropy(P_vec)
}
```

Handling of Enum variables

Enum variables need special handling as they are already converted to Booleans in the model. This is done by considering an Enum variable as a set of related Boolean variables. Thus while varying the observables, if a particular observable that has been converted from Enum to Boolean is fed to the diagnosis engine, then whole set of related Boolean variables is fed to the diagnosis engine.

5.2.3 Results

5.2.3.1 Results for Polycell Model

In this section we apply the two accuracy definitions to the polycell model which is described in section 2.2.1.

![Accuracy Vs. Entropy Graph](image)

Figure 5.2: Accuracy Definition 1 - Polycell (Accuracy Vs. Entropy)

Figure 5.2 shows the Accuracy Vs. Entropy graph for polycell model with no faults, using accuracy definition 1 in section 5.1. The accuracy is almost 1 (100%) when no fault is injected into the simulator. We calculate the accuracy and entropy for every possible input variable combination. Thus when the
system is completely healthy, as we vary the input variables, accuracy remains constant and maximum. The entropy is varying due to varying probabilities of the diagnosis at every position in the diagnosis array.

Figure 5.3: Accuracy Definition 1 - Polycell (Accuracy Vs. Entropy)

Figure 5.3 shows the Accuracy Vs. Entropy graph for polycell model with all single faults, using accuracy definition 1 in section 5.1. The accuracy is between 0.8 (80%) and 1 (100%) for every possible input variable combination with all combinations of single injected faults. Thus as we inject a fault, the accuracy decreases.

In order to correctly inject the fault influence, screened input should be fed to the simulator, so that it will give output which would be faulty. This faulty output along with input should be fed to the diagnostic engine so that fault is detected. Thus if inputs relevant to fault are not injected, the fault wont be detected. Hence special care should be taken for input and health vector combination that is fed to the simulator.

But since we are using all random inputs for a particular health vector, it may be possible that the fault is not propagated and outputs are not affected by the fault. This is the reason behind the decrease in the accuracy as we increase the number of injected faults.

Figure 5.4 shows the Accuracy Vs. Entropy graph for polycell model with all double faults, using accuracy definition 1 in section 5.1. The accuracy is between 0.4 (40%) and 0.9 (90%) for every possible input variable combination with all combinations of two injected faults. Thus as we inject another extra fault.

Figure 5.5 shows the Accuracy Vs. Entropy graph for polycell model with no faults, using accuracy definition 2 in section 5.1. The accuracy is exactly
Figure 5.4: Accuracy Definition 1 - Polycell (Accuracy Vs. Entropy)

Figure 5.5: Accuracy Definition 2 - Polycell (Accuracy Vs. Entropy)

1 (100%) when no fault is injected into the simulator. If compared with the figure 5.2, this definition is more exact and gives accuracy exactly equal to 1 (100%).

Figure 5.6 shows the Accuracy Vs. Entropy graph for polycell model with all single faults, using accuracy definition 2 in section 5.1. The accuracy is either 0.8 (80%) or 1 (100%) for every possible input variable combination with all combinations of single injected faults. Thus as we inject a fault, the accuracy fluctuates.

Figure 5.7 shows the Accuracy Vs. Entropy graph for polycell model with
all double faults, using accuracy definition 1 in section 5.1. The accuracy is between 0.4 (40%) and 0.9 (90%) for every possible input variable combination with all combinations of two injected faults. Thus as we inject another extra fault, the accuracy decreases significantly.
5.2. SENSOR PLACEMENT

5.2.3.2 Results for E-pin Model

Figure 5.8 shows the Accuracy Vs. Entropy graph for E-pin model with no faults, using accuracy definition 1 in section 5.1. The accuracy is almost 1 (100%) when no fault is injected into the simulator. We calculate the accuracy and entropy for every possible input variable combination. Thus when the system is completely healthy, as we vary the input variables, accuracy remains constant and maximum. The entropy is varying due to varying probabilities of the diagnosis at every position in the diagnosis array.

Figure 5.9: Accuracy Definition 1 - E-pin (Accuracy Vs. Entropy)
Figure 5.9 shows the Accuracy Vs. Entropy graph for E-pin model with all single faults, using accuracy definition 1 in section 5.1. The accuracy is between 0.8 (80%) and 1 (100%) for every possible input variable combination with all combinations of single injected faults. Thus as we inject a fault, the accuracy decreases.

Figure 5.10 shows the Accuracy Vs. Entropy graph for E-pin model with all double faults, using accuracy definition 1 in section 5.1. The accuracy is between 0.38 (38%) and 0.8 (80%) for every possible input variable combination with all combinations of two injected faults. Thus as we inject another extra fault there is a dip in the accuracy value.

Figure 5.11 shows the Accuracy Vs. Entropy graph for E-pin model with no faults, using accuracy definition 2 in section 5.1. The accuracy is exactly 1 (100%) when no fault is injected into the simulator.

Figure 5.12 shows the Accuracy Vs. Entropy graph for E-pin model with all single faults, using accuracy definition 2 in section 5.1. The accuracy is between 0.8 (80%) and 1 (100%) for every possible input variable combination with all combinations of single injected faults. Thus as we inject a fault, the accuracy decreases.

Figure 5.13 shows the Accuracy Vs. Entropy graph for E-pin model with all double faults, using accuracy definition 2 in section 5.1. The accuracy is between 0.3 (30%) and 0.6 (60%) for every possible input variable combination with all combinations of two injected faults. Thus as we inject another extra fault there is a dip in the accuracy value.

5.2.3.3 Accuracy Vs. Relative Observability for E-pin Model
In this section, we calculate the accuracy as a function of relative observability.
Figure 5.11: Accuracy Definition 2 - E-pin (Accuracy Vs. Entropy)

Figure 5.12: Accuracy Definition 2 - E-pin (Accuracy Vs. Entropy)

Figure 5.14 shows the accuracy as a function of observability for accuracy definition 1 described in section 5.1. When no faults are injected into the system, the accuracy of the diagnosis is almost constant and maximum for any number of observables fed to the diagnosis engine. But for single and double fault injection, the accuracy is very low for low relative observability and it increases with the increase in number of observables fed to the diagnosis engine. The maximum accuracy that can be achieved for single and double faults in the system is around 0.92 (92%) and 0.8 (80%) respectively. We take
an average of the overall accuracy for all combinations of single and double fault health vectors. Hence there are some cases where random insertion of the health vector and randomization of the observables is unrelated to the actual inserted fault. Thus maximum accuracy cannot be achieved in such cases.

Figure 5.15 shows the same result, but for accuracy definition 2 described in section 5.1.
5.2. SENSOR PLACEMENT

5.2.3.4 How to Increase the Accuracy?

Wise selection of the observables and hence placement of the sensors can increase the accuracy of the diagnosis significantly. In the E-pin case, we know that the *E-pin down sensor* and the *Exchange sensor* play a key role in detecting the E-pin error. Thus if we feed the inputs that drive these sensors to the simulator and also extract the observables that indicate the values of these two sensors and feed them to the diagnosis engine, the accuracy of the diagnosis increases to 100%. Thus, E-pin case can be solved accurately given that the values of the E-pin down sensor and the Exchange sensor are known all the time.

Since the E-pin case consists of relatively less number of components, it was easy to point out the significant sensors in the system. But this might not be the case for very complex systems and in such cases a certain heuristic is needed to point out the significant placement of the sensors.

Figure 5.15: Accuracy Definition 1 - E-pins (Accuracy Vs. Relative Observability)
Conclusions

6

6.1 Thesis Summary

In this thesis we address two main issues,

• (Semi-)automatic Lydia model generation from the Electrical layout designs that are drawn in Viewdraw software.
  In this section, we develop a tool that extracts the structure of the Lydia model from the Electrical layouts and inserts the behavior of the nets automatically. The user has to manually insert the behavior of the components in the model.

• Evaluation of the performance of the diagnosis using different accuracy definitions and entropy.
  In this section, we develop an experimental environment which compares the output from the diagnosis engine with the inserted health of the simulated system so that we can evaluate the accuracy of the diagnosis. In the process, we also calculate the entropy which can be defined as the diagnostic uncertainty. We also calculate accuracy as a function of relative observability.

We solve E-pin case study to drive our experiments and carry out a complete model-based diagnosis cycle. We extract the Lydia model framework from the model extraction tool and fill in the behavior manually. Based on the real time observations from the ASML board files, we diagnose the E-pin error in the system. We demonstrate that this diagnosis is equivalent to the diagnosis suggested by the same diagnosis engine for a Lydia model inferred by a human expert. Furthermore, we use the same model for carrying out the accuracy experiments. Thus we present a complete model-based diagnosis cycle in industrial domain.

6.2 Conclusions

• It is possible to derive the Lydia model framework from the Electrical layout designs. It is also possible to extract the minimal behavior of the netlists with \( n \) nodes. This is especially important in industrial case studies (like those in ASML) where the diagnosis experts know the importance of model-based diagnosis but avoid building models due to time constraints. Another reason for them to avoid building models is the
fact that the existing diagnosis methodologies work for them. But such methodologies are dependent on the system experts almost to all extent. *Using Lydia model framework extraction tool* and inserting behavior of specific components, model generation process becomes much more simpler.

- Behavior of the lower level components in the E-layout has to be inserted manually after the model framework is extracted.
- The *Lydia model framework extraction tool* is E-layout independent and hence can be used for any complex E-layout with any number of components.
- The structural accuracy of the Lydia model improves by extracting the framework from the Electrical layouts. This is inferred from the fact that the diagnosis of the E-pin error for the E-pin Lydia model extracted from the E-layout is equivalent to the diagnosis of the E-pin error for the E-pin Lydia model inferred by the human expert.
- By placing the sensors intelligently in the system, it is possible to increase the diagnosis accuracy. This also decreases the entropy (uncertainty) of the diagnosis. But such intelligent sensor placement becomes difficult as the complexity of the system increases.
- The accuracy of the diagnosis increases with increase in number of observables. Accuracy is highest for faultless health vector and average accuracy gets lower as the number of faults in the health vector start increasing, for varying observables.

### 6.3 Future Work

- Currently, the *Lydia model framework extraction tool* can extract the behavior of nets/buses in the E-layouts. The tool can be further extended to infer behavior of various low components. But this needs access to the E-layouts of the low level components. Thus, more fault models can be related to the model framework.
- We evaluate the diagnosis performance using two different accuracy definitions as explained in section 5.1. But the experimentation does not show significant differences in both the definitions and hence it is not possible to suggest the best definition for diagnosis accuracy. More cases should be solved and evaluated to find the differences between the two diagnosis definitions.
- Diagnosis performance can be measured considering a broader view with parameters like probability of injected health vector and time of diagnosis. For time variation evaluation, case studies with the necessity of modeling of time variation should be solved.
Bibliography

A grammar is a finite set of rules that specifies a language. ASCII files for ViewDraw are generated in formal language. Hence it is possible to describe each ASCII file with a context-free grammar. In context-free grammars, the left-hand side of the rewrite rule consists of a single nonterminal symbol.

BNF (Backus Naur Form) notation is used to describe the grammar. Following meta-symbols of BNF are used for grammar description of ASCII ViewDraw files.

- `::=` meaning "is defined as"
- `|` meaning "or"
- `<>` used to surround syntax rule names (non-terminal symbols)

These are supplemented by following Extended BNF (EBNF) meta-symbols.

- `*` meaning that terminal or non-terminal symbols can be repeated any number of times (and possibly be skipped altogether)
- `+` meaning that terminal or non-terminal symbols can appear one or more times
- `()` used to make grouping

```plaintext
GRAMMAR -

<E_layout> ::= <Version>
           <ViewDrawInternal>
           <BlockType>
           <Box>
           <SheetType>
           {
             {
               <Net> | <Bus>
               <Joints>*
           }
```
<Segments>+ 
)*

( 
<Symbol> 
<Attribute>* 
<AttachedPins>+ 
<UnattachedPins>+ 
)+ 
<E_layout>* 
<Label>* 
<Line>* 
<StringText>* 
<Arc>* 
<Circle>* 
<Comment>* 
<Attribute>* 
<Color>* 
<Pin>* 
<UnattachedAttr>* 

)+ 
<EndFile>

<Version> ::= V <VersionNumber> 
<VersionNumber> ::= <Integer> 

<ViewDrawInternal> ::= K <InternalId> <FileName> 
<InternalId> ::= <Integer> 
<FileName> ::= <Word> 

<BlockType> ::= Y <composite>|<Module>|<Unused>|<Annotate>|<Pin> 
<composite> ::= 0 
<Module> ::= 1 
<Unused> ::= 2 
<Annotate> ::= 3 
<Pin> ::= 4 

<Box> ::= D <LowerLeftX> <LowerLeftY> <UpperRightX> <UpperRightY> 
<LowerLeftX> ::= <Integer> 
<LowerLeftY> ::= <Integer>
<UpperRightX> ::= <Integer>
<UpperRightY> ::= <Integer>

<SHEET> ::= Z <ASIZE>|<BSIZE>|<CSIZE>|<DSIZE>|<ESIZE>|<A4SIZE>
                   |<A3SIZE>|<A2SIZE>|<A1SIZE>|<A0SIZE>|<FREESIZE>

<ASIZE> ::= 0
<BSIZE> ::= 1
<CSIZE> ::= 2
<DSIZE> ::= 3
<ESIZE> ::= 4
<A4SIZE> ::= 5
<A3SIZE> ::= 6
<A2SIZE> ::= 7
<A1SIZE> ::= 8
<A0SIZE> ::= 9
<FREESIZE> ::= 10

<Net> ::= N <NetId>
<NetId> ::= <Integer>

<Bus> ::= B <LowJointID> <HighJointID>

<LowJointID> ::= <Integer>
<HighJointID> ::= <Integer>

<Joints> ::= J <xPosition> <yPosition> <AloneJoint>|<DanglingJoint|
               <OnePinOneSegment>|<CongorTwoSegments>|
               <StTwoSeg>|<Solder>|
               <BusAloneJoint>|
               <BusDanglingJoint>|
               <BusOnePinOneSegment>|
               <BusCongorTwoSegments>|
               <BusStTwoSeg>|
               <BusSolder>

<xPosition> ::= <Integer>
<yPosition> ::= <Integer>
<AloneJoint> ::= 0
<DanglingJoint> ::= 1
<OnePinOneSeg> ::= 2
<CornerTwoSeg> ::= 3
<StTwoSeg> ::= 4
<Solder> ::= 5
<BusAloneJoint> ::= 6
<BusDanglingJoint> ::= 7
<BusOnePinOneSeg> ::= 8
<BusCornerTwoSeg> ::= 9
<BusStTwoSeg> ::= 10
<BusSolder> ::= 11
<Segments> ::= S <LowJointID> <HighJointID>

<LowJointID> ::= <Integer>

<HighJointID> ::= <Integer>

<Symbol> ::= I <Id> <SymbolName> <Sheet> <xLocation> <yLocation>
            <Orientation> <Scale> <Refdes>

@Id> ::= <Integer>

<SymbolName> ::= <Word>:<Word>

<Sheet> ::= <Integer>

<xLocation> ::= <Integer>

<yLocation> ::= <Integer>

<Orientation> ::= <Normal>|<90Degree>|<180Degree>|<270Degree>|<MirrorX>|<MirrorX90>|<MirrorX180>|<MirrorX270>

<Normal> ::= 0

<90Degree> ::= 1

<180Degree> ::= 2

<270Degree> ::= 3

<MirrorX> ::= 4

<MirrorX90> ::= 5

<MirrorX180> ::= 6

<MirrorX270> ::= 7

<Scale> ::= <Integer>

<Refdes> ::= "'"

<Attribute> ::= A <xLocation> <yLocation> <Size> <Orientation>
               <Origin> <Visibility> <String>

<Size> ::= <Integer>

<Origin> ::= <UpperLeft>|<MiddleLeft>|<LowerLeft>|<UpperCenter>|<MiddleCenter>|<LowerCenter>|<UpperRight>|<MiddleRight>|<LowerRight>

<UpperLeft> ::= 1

<MiddleLeft> ::= 2

<LowerLeft> ::= 3

<UpperCenter> ::= 4

<MiddleCenter> ::= 5

<LowerCenter> ::= 6

<UpperRight> ::= 7

<MiddleRight> ::= 8

<LowerRight> ::= 9

<Visibility> ::= <Invisible>|<Visible>|<AtrVisible>|<ValueVisible>

>Invisible> ::= 0

<Visible> ::= 1

<AtrVisible> ::= 2

<ValueVisible> ::= 3

<String> ::= <Word>
<AttachedPins> ::= C <NetId> <JointId> <PinId> <PinNum>
<NetId> ::= <Integer>
<JointId> ::= <Integer>
<PinId> ::= <Integer>
<PinNum> ::= <Integer>

<UnattachedPins> ::= X <PinId> <PinNum>

<Label> ::= L <xLocation> <yLocation> <Size> <Orientation>
           <Origin> <Scope> <LabelVisibility> <Inversion> <String>
<Scope> ::= <Local>|<Global>
<Local> ::= 0
<Global> ::= 1
<LabelVisibility> ::= <Invisible>|<Visible>
<Inversion> ::= <NotInverted>|<Inverted>
<NotInverted> ::= 0
<Inverted> ::= 1

(Line) ::= l (<xLocation> <yLocation>)+

<StringText> ::= T <xLocation> <yLocation> <Size> <Orientation>
                 <Origin> <String>

<Arc> ::= a (<xLocation> <yLocation>) (<xLocation> <yLocation>)
              (<xLocation> <yLocation>)

<Circle> ::= c <xLocation> <yLocation> <Radius>
<RadioButton> ::= <Integer>

<Comment> ::= "|" <Word>*

<Color> ::= Q <Colors> <FillStyle> <Line_Style>
<Colors> ::= 0|1|2|...|13|14|15
<FillStyle> ::= 0|1|2|...|13|14|15
<Line_Style> ::= 0

<Pin> ::= P <Id> (<xLocation> <yLocation>) (<xLocation>
            <yLocation>) <Type> <Side> <Inversion>
>Type ::= 0
<Side> ::= <Top>|<Bottom>|<Left>|<Right>
<Top> ::= 0
<Bottom> ::= 1
<Left> ::= 2
<Right> ::= 3

<UnattachedAttr> ::= U <xLocation> <yLocation> <Size> <Orientation>
                  <Origin> <Visibility> <String>

<Word> ::= <character>+ 

<Integer> ::= <Digit> | <Digit> <Integer>

<Digit> ::= 0|1|2|3|4|5|6|7|8|9

<character> ::= A|B|C.......Y|Z|a|b|c.......y|z|#|?|:|.|_|-|+|=|<Digit>
In this model, we insert the behavior for the components like sensor (DW_SENS_VSO_IND_2W) and motor (DW_ACT_MOTOR_DC_COIL) only. The other components are part of local sensor board and hence do not play any role in diagnosis of E-pin error. Hence the behavior of them is not manually inserted. We insert real_behavior and real_position systems to make the model similar to the one inferred by the human expert. Other behavior inserted in the model can be seen at the end of the model after the comments which state specifically.

```
system DW_SENS_VSO_IND_2W (  
      bool OUT ,  
      bool OUTR ,  
      bool h_DW_SENS_VSO_IND_2W  
)  
{
      probability (h_DW_SENS_VSO_IND_2W = true) = 0.99;
      h_DW_SENS_VSO_IND_2W => (OUT = real_out)
}

system real_behavior (  
      bool real_out,  
      bool real_up,  
      bool real_down,  
      bool real_middle,  
      bool h_real_behavior  
)  
{
      probability (h_real_behavior = true) = 0.99;
      h_real_behavior => (real_out =
                          (real_up and not(real_down) and not(real_middle)) or
                          (not(real_up) and real_down and not(real_middle)) or
                          (not(real_up) and not(real_down) and real_middle))
}
APPENDIX B. E-PIN LYDIA MODEL

system real_position (  
    bool real_pos,  
    bool real_true,  
    bool real_false,  
    bool h_real_position  
)  
{
    probability (h_real_position = true) = 0.99;
    h_real_position => (real_pos = real_true and not(real_false) or not(real_true) and real_false)
}

system DW_ACT_MOTOR_DC_COIL (  
    bool MOTR ,  
    bool MOT ,  
    bool h_DW_ACT_MOTOR_DC_COIL  
)  
{
    probability (h_DW_ACT_MOTOR_DC_COIL = true) = 0.99;
    h_DW_ACT_MOTOR_DC_COIL => (OUT = real_pos)
}

system DW_PCA_MIX_LSB2_AIUNBAL_17_24 (  
    bool AIUNBAL19 ,  
    bool AIUNBAL20 ,  
    bool AIUNBAL21 ,  
    bool AIUNBAL22 ,  
    bool AIUNBAL23 ,  
    bool EPINSN ,  
    bool EPINSP ,  
    bool GNDC54 ,  
    bool GNDC53 ,  
    bool GNDC52 ,  
    bool h_DW_PCA_MIX_LSB2_AIUNBAL_17_24  
)  
{
    probability (h_DW_PCA_MIX_LSB2_AIUNBAL_17_24 = true) = 0.99;
system DW_PCA_MIX_LSB2_AIBAL_1_8 (  
  bool AIBAL01N,  
  bool AIBAL01P,  
  bool AIBAL02N,  
  bool AIBAL02P,  
  bool AIBAL03N,  
  bool AIBAL03P,  
  bool AIBAL05N,  
  bool AIBAL05P,  
  bool AIBAL06N,  
  bool AIBAL06P,  
  bool AIBAL07N,  
  bool AIBAL07P,  
  bool AILVDTN,  
  bool AILVDTP,  
  bool AOLVDTN,  
  bool AOLVDTP,  
  bool h_DW_PCA_MIX_LSB2_AIBAL_1_8)  
{
  probability (h_DW_PCA_MIX_LSB2_AIBAL_1_8 = true) = 0.99;
}

system DW_PCA_MIX_LSB2_AIUNBAL_9_16 (  
  bool AIUNBAL09,  
  bool AIUNBAL10,  
  bool AIUNBAL11,  
  bool AIUNBAL12,  
  bool AIUNBAL13,  
  bool AIUNBAL14,  
  bool AIUNBAL15,  
  bool VREF_VSO,  
  bool GNDC45,  
  bool GNDC46,  
  bool GNDC51,  
  bool h_DW_PCA_MIX_LSB2_AIUNBAL_9_16)  
{
  probability (h_DW_PCA_MIX_LSB2_AIUNBAL_9_16 = true) = 0.99;
system net_2 (  
    bool node_1 ,  
    bool node_2 ,  
    bool h_net_2 )  
{
    probability (h_net_2 = true) = 0.99 ;
    
    bool connection ;
    h_net_2 => (connection = node_1 ) ;
    h_net_2 => (connection = node_2 ) ;
}

system net_4 (  
    bool node_1 ,  
    bool node_2 ,  
    bool node_3 ,  
    bool node_4 ,  
    bool h_net_4 )  
{
    probability (h_net_4 = true) = 0.99 ;
    
    bool connection ;
    h_net_4 => (connection = node_1 ) ;
    h_net_4 => (connection = node_2 ) ;
    h_net_4 => (connection = node_3 ) ;
    h_net_4 => (connection = node_4 ) ;
}

system main(  
    bool DW_SENS_VSO_IND_2W_2536_OUTR ,  
    bool DW_SENS_VSO_IND_2W_2536_OUT ,  
    bool DW_SENS_VSO_IND_2W_2535_OUTR ,  
    bool DW_SENS_VSO_IND_2W_2535_OUT ,  
    bool DW_ACT_MOTOR_DC_COIL_3210_MOT ,  
    bool DW_ACT_MOTOR_DC_COIL_3210_MOTR ,  
    bool DW_SENS_POS_LVDT_PRIM_SEC_2739_SECR ,  
    bool DW_SENS_POS_LVDT_PRIM_SEC_2739_SEC ,  
    bool DW_SENS_POS_LVDT_PRIM_SEC_2739_PRIMR ,  
    bool DW_SENS_POS_LVDT_PRIM_SEC_2739_PRIM ,
)}
bool DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_GNDC52,
bool DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_GNDC53,
bool DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_GNDC54,
bool DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_EPINSP,
bool DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_EPINSN,
bool DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL23,
bool DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL22,
bool DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL21,
bool DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL20,
bool DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL19,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AOLVDTNP,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AOLVDTN,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AILVDTPN,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AILVDTNP,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL07P,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL07N,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL06P,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL06N,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL05P,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL05N,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL03P,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL03N,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL02P,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL02N,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL01P,
bool DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL01N,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_GNDC51,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_GNDC46,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_GNDC45,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_VREF_VSO,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL15,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL14,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL13,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL12,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL11,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL10,
bool DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL09,
bool DWSENS_VSO_IND_2W_1561_OUTR,
bool DWSENS_VSO_IND_2W_1561_OUT,
bool h_DW_SENS_VSO_IND_2W_2536,
bool h_DW_SENS_VSO_IND_2W_2535,
bool h_DW_SENS_VSO_IND_2W_1561,
bool h_DW_ACT_MOTOR_DC_COIL_3210,
bool h_DW_SENS_POS_LVDT_PRIM_SEC_2739,
bool h_DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669,
bool h_DW_PCA_MIX_LSB2_AIBAL_1_8_2667 ,
bool h_DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671 ,
bool epin_error ,
bool h_net_2_1459 ,
bool h_net_4_1468 ,
bool h_net_2_3689 ,
bool h_net_2_1315 ,
bool h_net_2_1432 ,
bool h_net_2_1484 ,
bool h_net_2_1488 ,
bool h_net_2_1483 ,
bool h_net_2_1482 ,
bool h_net_2_1451 )
{

system DW_SENS_VSO_IND_2W
DW_SENS_VSO_IND_2W_2536 ;
DW_SENS_VSO_IND_2W_2536
( DW_SENS_VSO_IND_2W_2536_OUT ,
DW_SENS_VSO_IND_2W_2536_OUTR ,
h_DW_SENS_VSO_IND_2W_2536 );

system DW_SENS_VSO_IND_2W
DW_SENS_VSO_IND_2W_2535 ;
DW_SENS_VSO_IND_2W_2535
( DW_SENS_VSO_IND_2W_2535_OUT ,
DW_SENS_VSO_IND_2W_2535_OUTR ,
h_DW_SENS_VSO_IND_2W_2535 );

system DW_SENS_VSO_IND_2W
DW_SENS_VSO_IND_2W_1561 ;
DW_SENS_VSO_IND_2W_1561
( DW_SENS_VSO_IND_2W_1561_OUT ,
DW_SENS_VSO_IND_2W_1561_OUTR ,
h_DW_SENS_VSO_IND_2W_1561 );

system DW_ACT_MOTOR_DC_COIL
DW_ACT_MOTOR_DC_COIL_3210 ;
DW_ACT_MOTOR_DC_COIL_3210
( DW_ACT_MOTOR_DC_COIL_3210_MOT ,
DW_ACT_MOTOR_DC_COIL_3210_MOT ,
h_DW_ACT_MOTOR_DC_COIL_3210);

system DW_SENS_POS_LVDT_PRIM_SEC
  DW_SENS_POS_LVDT_PRIM_SEC_2739
  DW_SENS_POS_LVDT_PRIM_SEC_2739(  
    DW_SENS_POS_LVDT_PRIM_SEC_2739_PRIM,  
    DW_SENS_POS_LVDT_PRIM_SEC_2739_PRIMR,  
    DW_SENS_POS_LVDT_PRIM_SEC_2739_SEC,  
    DW_SENS_POS_LVDT_PRIM_SEC_2739_SECR,  
    h_DW_SENS_POS_LVDT_PRIM_SEC_2739 )

system DW_PCA_MIX_LSB2_AIUNBAL_17_24
  DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669
  DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669(  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL19,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL20,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL21,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL22,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL23,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL24,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL25,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL26,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL27,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL28,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL29,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL30,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL31,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL32,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL33,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL34,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL35,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL36,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL37,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL38,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL39,  
    DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_AIUNBAL40,  
    h_DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669 )

system DW_PCA_MIX_LSB2_AIBAL_1_8
  DW_PCA_MIX_LSB2_AIBAL_1_8_2667
  DW_PCA_MIX_LSB2_AIBAL_1_8_2667(  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL01N,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL01P,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL02N,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL02P,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL03N,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL03P,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL05N,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL05P,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL06N,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL06P,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL07N,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AIBAL07P,  
    DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AILVDTN,
APPENDIX B. E-PIN LYDIA MODEL

```
DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AILVDTP,
DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AOLVDTN,
DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AOLVDTP,
& DW_PCA_MIX_LSB2_AIBAL_1_8_2667

system DW_PCA_MIX_LSB2_AIUNBAL_9_16
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671;

DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671
( DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL09,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL10,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL11,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL12,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL13,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL14,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL15,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_VREF_VSO,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_GNDC45,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_GNDC46,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_GNDC51,
& DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671

system net_2 net_2_1459;

net_2_1459 ( DW_SENS_VSO_IND_2W_2536_OUTR,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL12,
& net_2_1459);

system net_4 net_4_1468;

net_4_1468 ( DW_SENS_VSO_IND_2W_2536_OUT,
DW_SENS_VSO_IND_2W_2535_OUT,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_VREF_VSO,
DW_SENS_VSO_IND_2W_1561_OUT, & net_4_1468);

system net_2 net_2_3689;

net_2_3689 ( DW_SENS_VSO_IND_2W_2535_OUTR,
DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL11,
& net_2_3689);

system net_2 net_2_1315;

net_2_1315 ( DW_ACT_MOTOR_DC_COIL_3210_MOT,
DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_EPINSN,
```
system net_2 net_2_1432 ;

net_2_1432 ( DW_ACT_MOTOR_DC_COIL_3210_MOTR, DW_PCA_MIX_LSB2_AIUNBAL_17_24_2669_EPINSP, h_net_2_1432 );

system net_2 net_2_1484 ;

net_2_1484 ( DW_SENS_POS_LVDT_PRIM_SEC_2739_SEC, DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AILVDTP, h_net_2_1484 );

system net_2 net_2_1488 ;

net_2_1488 ( DW_SENS_POS_LVDT_PRIM_SEC_2739_SEC, DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AILVDTP, h_net_2_1488 );

system net_2 net_2_1483 ;

net_2_1483 ( DW_SENS_POS_LVDT_PRIM_SEC_2739_PRIM, DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AOLVDTP, h_net_2_1483 );

system net_2 net_2_1482 ;

net_2_1482 ( DW_SENS_POS_LVDT_PRIM_SEC_2739_PRIM, DW_PCA_MIX_LSB2_AIBAL_1_8_2667_AOLVDTP, h_net_2_1482 );

system net_2 net_2_1451 ;

net_2_1451 ( DW_PCA_MIX_LSB2_AIUNBAL_9_16_2671_AIUNBAL13, DW_SENS_VSO_IND_2W_1561_OUTR, h_net_2_1451 );

//MANUALLY INSERTED BEHAVIOR

h_DW_SENS_VSO_IND_2W_2536 => ( epin_error = not(DW_SENS_VSO_IND_2W_1561_OUTR) and
    not(DW_SENS_VSO_IND_2W_2536_OUTR) and real_out = real_down);

h_DW_SENS_VSO_IND_2W_1561 => (
APPENDIX B. E-PIN LYDIA MODEL

\[
\text{DW\_PCA\_MIX\_LSB2\_AIUNBAL\_17\_24\_2669\_EPINSP} = \\
\text{DW\_PCA\_MIX\_LSB2\_AIUNBAL\_9\_16\_2671\_AIUNBAL13)} \\
\text{and (epin\_error} = \text{not(DW\_SENS\_VSO\_IND\_2W\_1561\_OUTR)} \\
\text{and not(DW\_SENS\_VSO\_IND\_2W\_2536\_OUTR));}
\]

\[
h_{\text{DW\_ACT\_MOTOR\_DC\_COIL\_3210}} \Rightarrow \\
(\text{DW\_PCA\_MIX\_LSB2\_AIUNBAL\_17\_24\_2669\_EPINSP} = \\
\text{DW\_PCA\_MIX\_LSB2\_AIUNBAL\_9\_16\_2671\_AIUNBAL13 and real\_pos} = \text{real\_true});
\]