Fault Tolerant Analysis of Automotive Systems

Thesis submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Technology
in
Computer Science and Engineering
by

Aurosish Mishra
06CS3027

Under the supervision of:
Prof. Partha Pratim Chakrabarti

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
West Bengal, India, 721302

May 10, 2010

CERTIFICATE

This is to certify that the thesis entitled "**Fault Tolerant Analysis of Automotive Systems**" is a bonafide record of authentic work carried out by **Aurosish Mishra** under my guidance and supervision for the fulfillment of the requirement for the award of the Bachelor of Technology (Honours) degree in **Computer Science and Engineering** at Indian Institute of Technology, Kharagpur. The work incorporated in this, has not been, to the best of my knowldege, submitted to any other University or Institute for the award of any degree or diploma.

Prof. Partha Pratim Chakrabarti
Department of Computer Science and Engineering
IIT Kharagpur

ACKNOWLEDGEMENT

I would like to express my gratitude towards Prof. Partha Pratim Chakrabarti, Department of Computer Science and Engineering, IIT Kharagpur, for extending all the necessary support throughout the duration of the project and for being a constant source of inspiration. Taking time out of his busy schedule, he ensured that the project work was carried out methodically and meticulously. I am greatly indebted to him for all his creative ideas and inspirations which gave us an impetus to come out with constructive creations.

I would also like to express my gratitude to Mr. Satya Gautam Vadlamudi for his intellectual guidance, constant attention and unceasing encouragement during the research and development of this project.

I would also like to avail this opportunity to express my sincere gratitude to my parents and my brother who have been an endless source of encouragement for me.

Aurosish Mishra
06CS3027
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
May 10, 2010

Contents

Abstract

Over the years, Fault Tolerance has become the most sought-after requirement in high-availability or life-critical systems. Fault-tolerant electronic sub-systems are becoming a standard requirement in the automotive industrial sector as electronics becomes pervasive in present cars. The existing fault-tolerance analysis approach can handle quality faults emanating from loss of precision in signal values of automotive systems, in addition to logical faults. This analysis can aid the designer to identify where and how precision losses occur in both software and hardware components. This analysis is performed on operation-level (Simulink) models, and hence provides early detection of weak spots in the system.

The existing approach uses two fault analysis methods in tandem: a fault-injection and simulation method is used which evaluates the quality degradation of output signals of the system, under specific testcases and fault-scenarios; and a look-up table based approach, in which quality degradation of the complete system is obtained by a series of table lookups, which store the characterized quality degradation behavior of individual components. The LUT based method performs a quick but approximate analysis and yields better coverage with respect to quality behavior.

The exhaustive analysis of quality fault-tolerance is a computationally intensive problem. So, we focus on improving the LUT based approach for performing fault tolerant analysis. In this thesis, we present a novel technique to perform characterization of components using multiple functional testcases that are quite varied and hence capture most of the component behaviour in various faulty scenarios. Improving the characterization methodology, leads to improved quality LUTs for each component, using which dynamic analysis produces much better results. We use our approach to characterize two systems, namely a Fault Tolerant Fuel Controller and an Anti-lock Braking System. Apart from this, we also suggest techniques for performing compaction of LUTs to satisfy time and storage requirements.

Chapter 1

Introduction

Fault-tolerant electronic sub-systems are becoming a standard requirement in the automotive industrial sector as electronics becomes pervasive in present cars. Fault-tolerance or graceful degradation is the property that enables a system (often computer-based) to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a navely-designed system in which even a small failure can cause total breakdown. Fault-tolerance is particularly sought-after in high-availability or life-critical systems.

The basic characteristics of fault tolerance require the following:

1. No single point of repair

2. Fault isolation to the failing component

3. Fault containment to prevent propagation of the failure

4. Availability of reversion modes

In addition, fault tolerant systems are characterized in terms of both planned service outages and unplanned service outages. These are usually measured at the application level and not just at a hardware level. The figure of merit is called availability and is expressed as a percentage. For example, a five nines system would statistically provide 99.999% availability. Fault-tolerant systems are typically based on the concept of redundancy.

1.1 Motivation

With the large scale proliferation of electronics and software as building blocks of automotive systems, fault- tolerance has emerged as a first class design requirement. It is of paramount importance to develop systems which preserve functionality in-spite of failures and errors in electronic, communication and processing components. This requirement stems from the fact that unlike mechanical components, which have graceful degradation in case of failures, failure of electronic components causes sharp changes in system behavior. For example, a defective mechanical steering column may cause some variations in the steering torque provided by it, but a stuck-at-fault in a processor output providing signals to a steer-by-wire system may cause dramatic increase or decrease in steering torque. Additionally, automotive systems are safety-critical systems which must conform to stringent industry safety norms for fault-tolerance. Hence, it is important to design systems which are provably resilient to faults which may emanate from various sources and causes.

Failure of electrical and electronic components due to defects and aging is a well documented phenomenon [16]. Chips, sensors, power supplies, and electromechanical actuators are known to fail either permanently, or for short time durations (transient failures), or by losing precision (behavior of measurable quantities like output voltage varies from fault-free behavior). Additionally, 'bugs' in software and even in the design of the hardware multiprocessor platform, may cause transient and permanent failures. These failures manifest themselves as errors in the output of the control system, and then the actuator. These errors follow a feedback loop to the sensors of the automotive system by propagating across the mechanical components of the plant. This may further deviate the behavior of the automotive system from the fault-free case.

The analysis of precision loss of automotive signals (analog, digital and data in software) is an important but insufficiently addressed issue in the design-flow of automotive control systems. Several automotive system components, like sensors, software and hardware blocks, introduce sporadic quality faults ranging from shift in signal trajectory (for example bias faults in sensors), to erroneous outputs for a short time duration (missing data from a camera, or software bugs). These quality faults do not necessarily render the affected signals useless, but cause a loss of precision. The precision loss introduced by one component propagates across other fault-free as well as faulty components resulting in aggregation of precision losses with each propagation, and with the elapse of time. In order to tackle these problems, several fault-tolerance (FT) mechanisms are employed by designers. An analysis framework, for systematically evaluating different fault-tolerant automotive system design options, for quality faults is an important ingredient of any automotive system design-flow. This research strives to address the quality fault analysis problem for automotive systems.

1.2 Background

In order to design fault-tolerant systems, a gamut of techniques and design artifacts are incorporated in various components of the system. These design features enable the functioning of the system in the presence of environment induced failures and aging failures. The most common technique employed across software and hardware components of the automotive control system, as well as in sensors and actuators (with multiple windings), is replication [10]. A study of techniques for developing fault-tolerant processing-nodes using replication in different ways is presented in [2]. Similarly, in some cases software components are replicated on multiple processing-nodes to ensure that some replica of the software component is operational even in case of failure of a subset of processing-nodes [11]. Apart from replication, another technique for providing fault-tolerance in the software layer is embedding non-identical alternate ways for computing the value of a signal. Often look-up tables and time delay neural networks (TDNNs) [9] are used to approximate the value of certain signals. These allow the computation of the same signal by multiple sets of software functions. Another technique providing fault-tolerance to transient errors (like soft-errors [15] and state dependent software and hardware bugs) is rescheduling, wherein a task is re-executed if an error is detected [10].

Aside from fault-tolerant design techniques, two essential ingredients for achieving fault-tolerance are a diagnosis component and a reconfiguration or selection component. Diagnosis is required in order to detect the faulty components of the system and inform the reconfiguration mechanism of the same. Most diagnosis methods operate by recording the values of certain signals and checking values of these signals against pre-defined assertions or goldenized models. Reconfiguration consists of changing the topology of the software or hardware systems so as to overcome the failure of single or multiple components. In the simplest form reconfiguration involves switching the source of a particular signal from one component to another. More complex reconfiguration methods which employ different algorithms in response to the set of available resources (computing, sensory and actuators) [12] have also been explored by researchers.

While designing a fault-tolerant automotive system, the designer may want to perform a 'what if ?' analysis for different fault-tolerance strategies. Additionally, once a final design point is reached, the designer must analyze the design to assure that certain fault-tolerance requirements are satisfied. Clearly, a framework for analysis of fault-tolerance is an important requirement for fault-tolerant automotive system design.

Fault-tolerance analysis methods in the literature can be broadly classified as per the fault-tolerance requirements which must be satisfied by the system:

1. *Logical Analysis:* The analysis of logical system failures [10], and resilience under specific fault-scenarios (a sequence of occurrence of faults). Logical faults analyzed by these methods completely degrade the output of components, such that these output values cannot be utilized. A logical fault naturally arises due to hardware failures (failure of power supply, stuck-at-faults, aging) and bugs in the software.

2. *Quality Analysis:* The analysis of loss of precision in various components and the propagation of precision errors. There are several sources of quality faults, for example bugs in the software which are invoked in a few execution time-slots, resulting in erroneous outputs in only those execution time-slots. Moreover, quality faults may arise in hardware due to shift in supply voltage to sensors, noise in sensors and actuator components, and missing output data from sensors (like cameras). While analyzing quality faults, the propagation of quality degradation across fault-free as well as faulty, software, hardware and mechanical components must be studied. Additionally, it may be noted that in some cases diagnostic mechanisms monitor the quality degradation of

a subset of signals, and notify a warning when the tolerance limits are breached. In such cases, it is often of interest to analyze the extent of quality degradation of un-monitored signals when certain diagnostic mechanisms do not issue a warning.

3. *Reliability Analysis:* The analysis of mean time to failure (MTTF) for the system, given the MTTF of individual components [5]. This is especially useful for analyzing aging faults.

Currently, various dynamic (simulation-based) and static analysis methods are employed at various stages of the design-flow for fault-tolerance analysis. Simulation-based methods are employed on operation-level, as well as implementation-level models for logical and quality analysis. Typical operation-level models used are Simulink models [8], while C/C++ processor simulators and emulation setups executing automotive software are employed for implementation-level analysis. Model checking based methods may be employed for logical analysis of designs, while graph-based algorithms are employed to perform reliability analysis of automotive systems [5].

1.3 Objectives

The basic objective of the project is to:

Analyze the accuracy and coverage of quality-LUTs, and make necessary improvements, with a view to developing an elegant method for performing fault tolerant analysis of Automotive Systems.

The major part of the project focuses on developing a novel technique for characterizing components of an automotive demonstration,"Fault Tolerant Fuel Controller", provided by Simulink. The existing characterization approach leads to incomplete and somewhat inaccurate Look-up tables(LUTs). Ideally, we want our quality LUTs to be as accurate and as complete as possible so as to capture almost all faulty scenarios. Using the improved LUTs, we aim at performing dynamic and static analysis in order to obtain the counterexamples that violate the fault tolerance requirements. We also want to perform compaction of the LUTs to reduce simulation time and storage space. The proposed methodology is also tested on another model called the “Anti-lock Braking System“ so as to examine the advantages provided by this method.

1.4 Summary of Work Done

We analyze an existing approach which uses lookup table-based and fault-simulation-based methods in tandem to obtain good coverage, while maintaining the accuracy of analysis. We try to rectify the shortcomings of the exisiting approach to come up with a generic fault tolerant analysis approach.

We present a modified technique for improving the characterization of components of Automotive Systems. This involves developing an alternate representation to capture the behaviour of signals with minimal storage requirements. Improving the characterization methodology, leads to improved quality LUTs for each component, using which dynamic analysis produces much better results. We elucidate the advantages of the new methodology, by applying it to characterize two systems, provided by the Simulink automotive demonstrations:

- A Fault Tolerant Fuel Controller
- An Anti-Lock Braking System

The varied behaviour of different components makes it a challenging problem to characterize all components accurately and completely. The suggested method keeps the various fault scenarios that a component is susceptible to, under consideration. The functional testcases chosen for characterizing components have a wide variation in behaviour so as to capture as much faulty behaviour as possible. Apart from this, we also suggest a technique for performing compaction across the obtained LUTs so as to satisfy time and storage constraints.

1.5 Organization of Thesis

The remainder of the thesis is organized into several chapters as follows:

- *Chapter 2* presents the exisiting approach for performing fault tolerant analysis, which involves using a lookup table-based and a fault-simulation-based method in tandem. This chapter introduces the basic concepts of an operation level model, the used technique for characterization of components and the advantages of the Look-Up table based analysis method.

- *Chapter 3* presents the modified approach to characterize components of an automotive system, leading to generation of better quality LUTs. It also explains the needs for modifying the existing approach. Apart from this, we also present a technique for compaction of LUTs and the approaches for performing static and dynamic analysis.

- *Chapter 4* presents an account of all the work done, along with the observations accompanying the new characterization method. We present the characterization of two components of the Fault Tolerant Fuel Controller and one component of the Anti Lock Braking system to establish the efficacy of our method. We also present the improved dynamic analysis results obtained using the modified quality LUTs.

- *Chapter 5* outlines the conclusions of the project and prepares an action plan to be followed in the near future.

Chapter 2

Current Approach for Fault Tolerant Analysis

2.1 Operation Level Models

Operation-level modeling languages, for example the Simulink modeling language [8], provide a versatile framework for modeling various aspects and abstractions of automotive systems. These models are capable of representing not only function-level models of the automotive system, but also some details of the architectural platform on which the automotive system is executing (mapping to same processor, buffers and buses etc).

An operation-level model(M_{op}) [17] consists of operations(Op) which have a set of input(In_{Op}) and output ports(Out_{Op}), and signals(Sig) connecting different ports of these operations. Each operation corresponds to a functional component of the system, which may range from a software code-block to an analog component, or even a sensor. A discrete-event semantics is considered in this work, which is more pertinent since several operations are mapped to software components (operating on sampled signals in discrete steps). Each signal denotes the value which is updated by the 'source' operation in every time-slot. It may be noted that a signal is a virtual connection between operations and may correspond to physical quantities, like output voltage generated by a filter, or may correspond to a data value generated by a software block.

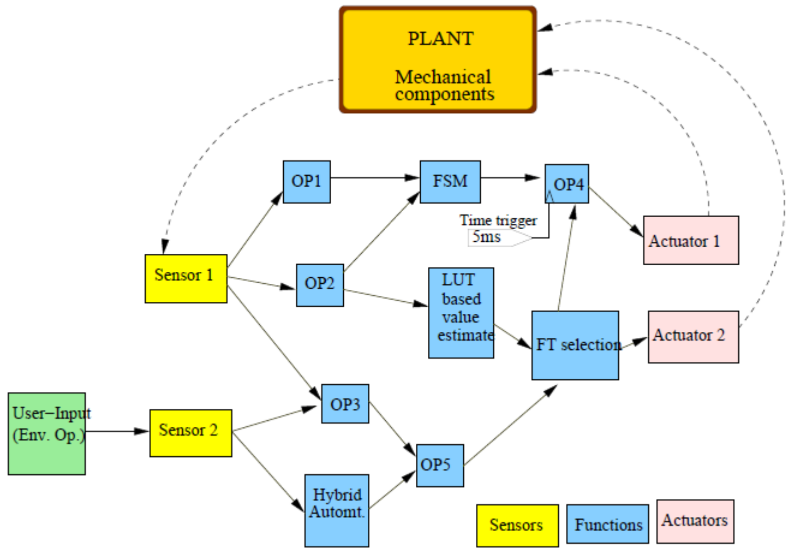

Figure 2.1: A Schematic Representation of an Operation-Level Model

The schematic of an operation-level model can be represented as a directed graph with operations corresponding to nodes and signals corresponding to edges ($Sig \subseteq Op \times Op$). It broadly consists of five sets of operations, namely, sensor operations, control operations, actuator operations, plant operations, and environment operations (Figure 2.1). Additionally, we note that signals connect sensors to the control, control to the actuators, actuators to the plant,and plant to the sensors, in a cyclic manner. Environment operations, for example, the 'time- trigger' and the 'user-input' in Figure 2.1, model the behavior of the driver and the environment, and do not have incoming signal edges. It is notable that removing signals between the plant and sensors (or actuators and the plant) causes the underlying graph structure to be a directed acyclic graph (DAG). We call this modified operation-level model, the open operation-level model, and the DAG structure is represented as $M_{open}(Op, Sig_{open})$, where $Sig_{open} \subset Sig$. Loops in the control component may be logically treated as being connected to a virtual actuator, an operation-free signal in the plant, and a virtual sensor (Figure 2.2). In this case, when the signals between the plant and sensors are removed, the graph structure of the operation-level model is a DAG.

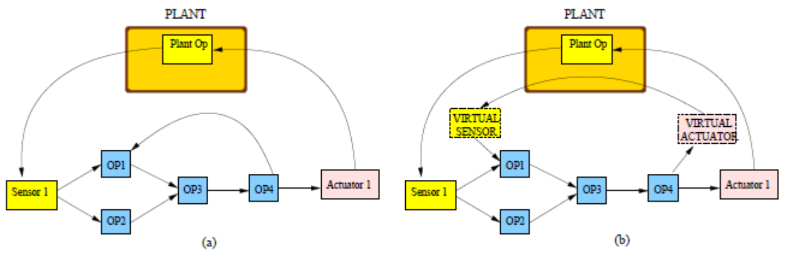

Figure 2.2: Handling loops in the automotive control as a virtual signal path through the plant. (a) Original loop in the control, (b) Loop through a virtual sensor, and virtual actuator.

An example schematic representation of an operation-level model, broadly similar to schematic representations of several operation-level models of automotive systems, is presented in Figure 2.1. Each operation in this model corresponds to either: a task of the control application, or a sensor operation (obtains value of certain parameters and user inputs), or an actuator, or components of the plant (consisting of mechanical parts). Each operation is represented by either a logical or arithmetic function, or a state-machine like a FSM, or hybrid I/O automata [13]. In most automotive systems of interest, the control system is almost entirely software based, thereby sensor outputs are immediately converted to data items provided as inputs to control software components. Moreover, many control (software) components may be time triggered, such that they start/resume execution at specific time instances. For example, the operation OP_4 executes only when 5 ms elapses from the start of execution, even if inputs are available earlier. Edges between operations denote virtual functional connection between them, which map the output trajectory of a source operation as the input trajectory for the destination operation.

2.2 Modelling Errors at Operational Level

One of the most important requirements of an operation-level fault-simulation framework is the modeling of various quality faults at operation-level abstraction. The origin of faults typically lie in circuit-level or assignment-level details of an implementation. For example, a soft error causes a transient bit-flip in a memory cell or register, or a software bug may cause incorrect values. It is important to abstract these faults to the level of operation-level models, while preserving the essence of the manner in which a fault affects a signal value. In this work we propose to abstract the effects of quality faults as either noise, shift, or spike faults (Figure 2.3). These can be utilized to abstract the effects of the following types of faults (among others) in operation-level models:

1. *sensor faults* leading to added noise on output and shift in output signal trajectory (noise and shift faults).

2. *missing data from sensors* leading to arbitrary sensor outputs (spikes) in certain time-slots (for example missing data from a camera).

3. *software bugs* and *hardware errors* which do not manifest themselves in every execution run can be viewed as spike faults in certain time-slots in which they are exercised. These spike faults overapproximate the defect introduced by the bug/error by producing the maximum/minimum value possible for the signal in the said time-slot.

4. *precision losses* in software components are modeled as trajectory shifts. These shifts can occur due to typecasting bugs, and due to porting control software to embedded automotive platforms which may not support many high-precision and floating-point operations.

5. *hardware recovered soft-errors* and *temporary logical faults* manifest as spikes (sudden and short change) on signals(spike fault).

It is assumed that permanent logical faults are detected by appropriate components in the hardware layer such that fault-silence can be implemented [2]. Additionally, operations are assumed to be instrumented such that they are fault-silent, in case any essential input signal indicates fault-silence. Hence, propagating the effects of permanent logical faults by fault-simulation is trivial. Such being the case, we shall henceforth focus only on quality faults in this work, while stating that the proposed methodologies are capable of handling logical faults as well.

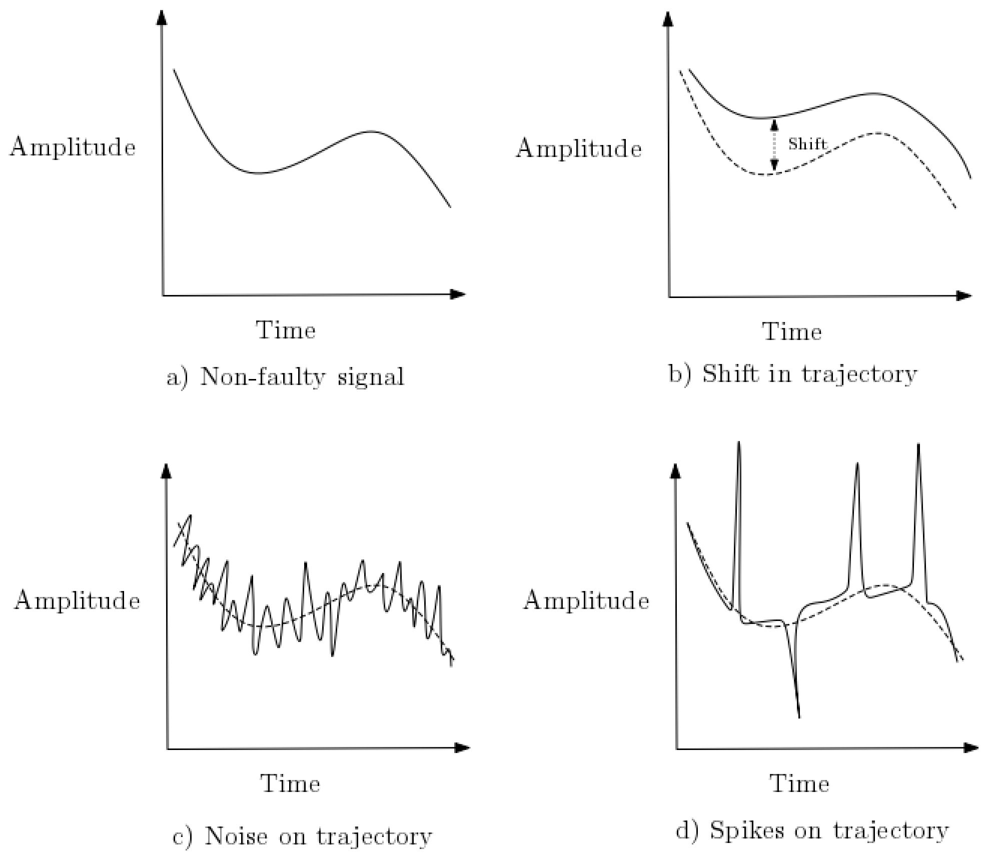

Figure 2.3: Modelling of Errors at the Operation Level

It may be noted that some of the aforementioned faults do not originate in software control components of the automotive system. However, due to fault propagation, their effects are still observed on the outputs of various software control components (among others). Hence it is imperative for any fault-tolerance analysis method to address the propagation of the aforementioned faults across different types of software, hardware, and mechanical (from the plant) components.

For modeling quality degradations, we associate each type of quality degradation with an appropriate measure for denoting the extent of quality degradation (the error). Measures of quality degradation for the three types of quality faults introduced earlier are presented next:

1. *Shift:* in signal trajectory, for which quality degradation is denoted by the maximum deviation in signal value among all time instances within the simulation window.

2. *Noise:* for which quality degradation is described by amplitude of the overlaying additive random noise signal

3. *Spikes:* for which the quality degradation is denoted by the number of spikes. The peak value of these spikes is bound by the upper limit of the operation-range or data-type (in case the signal is a digital data).

2.3 Simulation Based Analysis

2.3.1 Inputs for Fault Tolerant Analysis

There are three inputs to the simulation-based fault-tolerance analysis framework, namely *the testcase, the fault-scenario and the fault-tolerance requirement specifications.*

Testcases typically describe a set of typical as well as critical maneuvers (job sequences) which are performed by the automotive system. Additionally, testcases may also be generated to perform directed analysis of the automotive system. Usually, the sensor inputs which come from the user are modeled by these testcases. In case only a part of the system is being tested, certain signals from the 'plant', which were generated in response to control signals from some other sub-system, are also included in the test-suite.

The second input to the simulation-based fault-injection framework is the *description of the fault-scenario* under which the system must be analyzed. Fault-scenarios may be described explicitly by stating a set of faults which must occur. In the case of quality faults, in addition to the information of which faults occur, the measure of quality degradation must also be stated. Hence a fault-scenario is a set of triplets of the form: $<location, fault-type, measure>$, where location denotes the signal which is afflicted by the fault, while fault-type and measure denote the type and measure of error (measure is irrelevant for logical faults). For example, a fault-scenario may specify that 'at most 5 spikes (type and measure) may be introduced by all (location) software components'.

It must be noted that all types of faults cannot occur on every signal in the model. In this work we allow noise, shift and spike errors on sensor outputs, shift and spike errors on outputs of software components, spike errors on outputs of hardware components, and no errors on outputs of plant components (we argue that sensor errors can account for plant errors).

While specifying fault-scenarios as inputs to the analysis framework, it is important to account for the correlation between various faults. For example a common power supply induces several correlated noise and bias (shift) faults in all sensors it supplies power to. In this work, we assume that fault-scenarios are external inputs to the proposed analysis framework, and are defined by the designer by taking effects of correlation into consideration.

The third set of inputs to the simulation-based fault analysis framework is the *set of fault-tolerance requirements* which must be satisfied by the system. Logical and timing properties specifying the design intent [3] of the system can be used as specification for the fault-tolerance analysis step. Additionally, specific properties for checking bounds on quality degradation may be specified (for example an upper bound on amount of superimposed noise). Besides this, more involved properties may be written where acceptable quality degradation is a function of time [4].

In this work, we consider quality fault-tolerance requirements to be properties of the form $\bigwedge_i \psi_i \Rightarrow \phi$, where ψ_i is an antecedent proposition and ϕ is the consequent proposition. Both, the antecedent and the consequent propositions, are either True, False, or are propositions denoted by tuples of the form: $<Sig, fault-type, limit>$. Here each tuple specifies that the sum of quality degradations of type 'fault-type', at all the signals location $\in$ *Sig*, must be upper-bounded by *'limit'*. However, there is a crucial difference between the antecedent and consequent propositions. While the antecedent propositions reason about the additive quality degradations (like in case of fault-scenarios), the consequent proposition reasons about the quality degradation of the signal as compared to the golden (fault-free) signal.

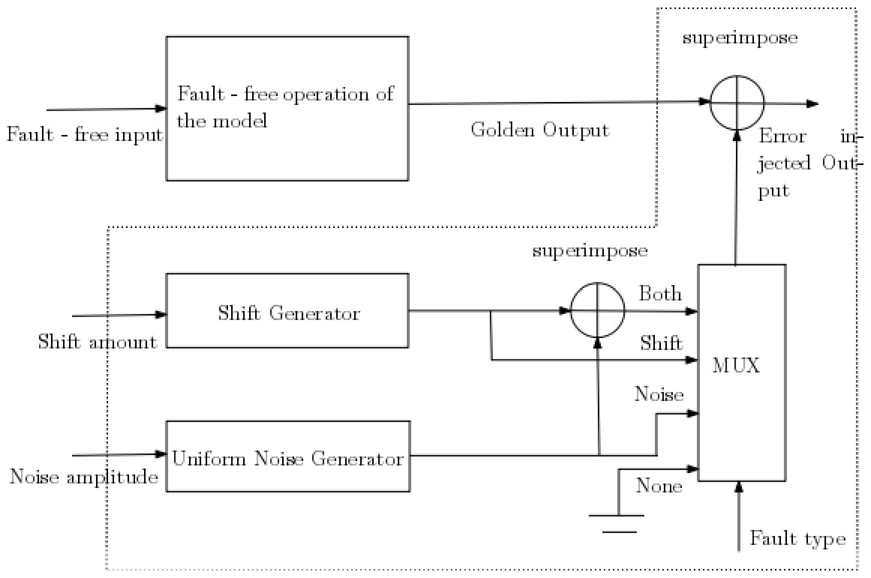

Figure 2.4: A Schematic Representation of Fault Injection

2.3.2 Fault Simulation Based Analysis

Given the three inputs to the fault-tolerance analysis framework, the simulation-based fault-tolerance mechanism consists of fault-injection, simulation of the operation-level model, and checking assertions corresponding to the fault-tolerance requirement properties. For this, first we instrument the native operation-level model of the automotive system with 'fault-injection' operations. These operations superimpose errors on the native input signal (which may be error free or may itself contain some prior acquired errors) of such operations. Here it may be noted that all the fault types proposed in this work are amenable to signal superposition.

An example fault-injection operation, implemented in Simulink, is illustrated in Figure 2.5. This operation can superimpose noise, shift, and spike errors (at most one spike) on the native signal. The amount of noise and shift, as well as the number of spikes (0 or 1 spike in this example), are fixed by assigning values to the parameters *'noise-amplitude'*, *'shift-amplitude'* and *'spikes0/1'*. For example, *'shift-amplitude'* set to 0 indicates the absence of a shift fault. The values which must be assigned to these parameters are obtained from elements of the fault-scenario, corresponding to the given sig-

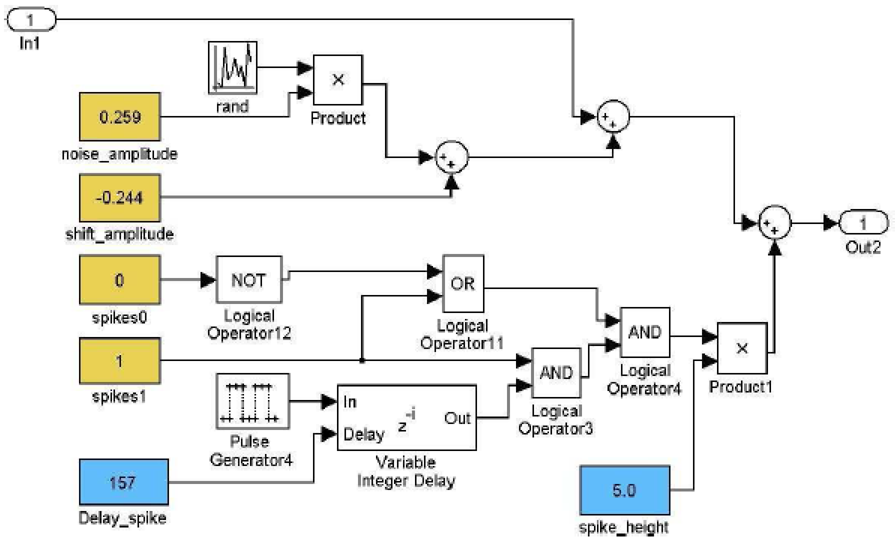

Figure 2.5: A Fault Injection mechanism for introducing errors on a signal

nal. Other parameters, like the height of a spike and the position (along the time-axis) of introduction of a spike are set, either randomly, or by the user after a study of the signal waveform.

Once the operation-level model of the automotive system has been instrumented with the 'fault-injection' operations, fault-simulation consists of checking fault-tolerance requirements for all relevant fault-scenarios, for each functional testcase. In each fault-simulation, monitored signals, which must conform to fault-tolerance requirements, are compared against logged signal trajectories of the corresponding fault-free run, for the given functional testcase. The difference in trajectory between the fault-injected and fault-free runs for each monitored signal is computed and logged. Thereafter, the logged difference trajectory is appropriately processed for checking shift, noise, and spikes causing deviation from the fault-free trajectory.

The difference between the fault-injected and fault-free (golden) signal trajectories are computed as the difference in amplitudes of the signals in different time slots. Once the *difference trajectory* is computed, the number of spikes is obtained as the number of time-slots in which the difference is greater than a limit (spike-limit) which depends on the native trajectory. For computing the amount of shift and noise errors, we consider windows of a certain number of time-slots starting at each time-slot within the simulation window. The amount of shift is then computed as the maximum, over all windows, of the average value in each difference trajectory window-segment (while ignoring spikes). This essentially tries to mimic the behavior of a column filter. The amount of noise is computed as the maximum amplitude difference within a window, amongst all windows.

Hence, we obtain a set of quality degradation tuples of the form: $<msignal, v_{noise}, v_{shift}, v_{spikes}>$, where v_{noise} , v_{shift} , and v_{spikes} are the quality degradations of different types (noise, shift, and spikes) for the monitored signal, $msignal$. Fault-tolerance requirements are checked by comparing the quality degradation tuples of the monitored signals, with their limits in the fault-tolerance requirement queries.

2.3.3 Complexity of Exhaustive Fault Simulation

An important aspect of the quality fault analysis framework is the notion of a complete set of fault-scenarios for the automotive system (for a given functional testcase). Since the measurement of quality degradation for noise and shift faults involves real numbers (with infinitely many possible values within a bounded interval), a finite representative set of real numbers may be selected by the designer for defining a complete set of fault-scenarios. Such being the case, let $\Pi_{noise}(l)$ and $\Pi_{shift}(l)$ be the finite sets of values of degradation for the additional noise and shift (respectively) introduced on signal l. Now consider a system where S_s, S_c , and S_h are sets of sensor, software, and hardware output signals (with $n_s = |S_s|, n_c = |S_c|, and n_h = |S_h|$). Additionally, let us assume that all queries have an antecedent proposition of the form $<S_s \cup S_c \cup S_h, spike, n_{spikes}>$, restricting the total number of injected spikes to at most n_{spikes}. Clearly, there are $\sum_{i=0 \ldots n_{spikes}} C_i^{n_s+n_c+n_h+i-1}$ fault-scenarios arising out of different ways in which spikes may be injected (selecting i signals from $n_s + n_c + n_h$ with repetition). Additionally, assuming that $|\Pi_{noise}(l)| = |\Pi_{shift}(l)| = n_{quant}$, the number of possible ways of occurrence of shift and noise faults at sensor outputs is $n^{2n_s}{}_{quant}$. Also, the number of possible ways of occurrence of shift faults at software component outputs is: ${n_{quant}}^{n_c}$. Hence, the total number of fault-scenarios possible for each testcase is:$(\sum_{i=0 \ldots n_{spikes}} C_i^{n_s+n_c+n_h+i-1}){n_{quant}}^{2n_s+n_c}$. For the automotive system case study introduced later,$n_s = 4, n_c = 20$ and $n_h = 3$. The number of fault-scenarios for $n_{quant} = 3$ and $n_{spike} = 5$, is approximately 4.6×10^{19}.

2.4 Look-Up Table Based Analysis of Operational Models

The analysis of quality degradation of automotive systems is a computationally intensive problem, often requiring a very large number of simulation runs, as illustrated in Section 2.3. Hence, it is paramount to devise methods to reduce the amount of computation required for each simulation run. In this work we characterize the quality response of components, and then utilize this result to perform the analysis for the complete system. Characterization is conducted on individual operations, or a user defined set of operations, to analyze their response to input signal quality degradation, as well as spikes on inputs. In the proposed lookup table-based method, the results of characterization are tabulated, and quality degradation of a system is computed by a series of table lookups.

2.4.1 Characterizing Operations

Let us consider an operation OP, which has k input signals $i_0, i_1, ..., i_{k-1}$ and m output signals $o_0, o_1, ..., o_{m-1}$. The first step for characterization involves logging the input and output signal trajectories for fault-free behavior, for each testcase $tc_i \in TC$. For the illustrated setup (in Simulink) for characterization shown in Figure 2.6, 'engine_speed.mat' and 'manifold_pressure.mat' are log files for the fault-free input signals, while 'pumping_constant_output.mat' is the log file of the output signal, for one testcase.

Characterization proceeds exactly like fault-injection and simulation as described earlier in Section 2.2. The fault-scenarios for characterization involve invoking noise, shift and spike faults on the input signals. Even though several types of errors are not injected on many input signals during full-system fault-simulation, characterization requires that all types of errors be injected on each input signal. This is because errors injected at input signals must not only model errors introduced by the immediate source operations, but also model propagated errors which originate beyond the source of the signal, from operations which could introduce any type of error (like sensors).

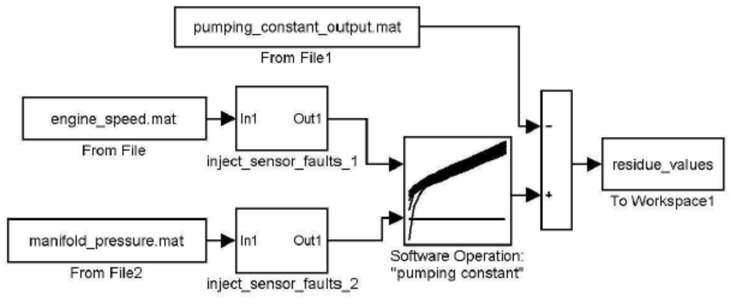

Figure 2.6: A setup for characterizing a two-input component

For an exhaustive characterization, all combinations of values of noise and shift faults at the user defined quantization-level (n_{quant} points each), and spike faults with an upper bound on the number of spikes (n_{spikes}), are invoked. Hence $({n^2}_{quant})^k$, fault-scenarios iterating over n_{quant} noise, and n_{quant} shift values for each of the k inputs must be considered. Additionally, $0...n_{spike}$ spikes are spread across the k inputs, contributing to a factor of $\sum_{i=0...n_{spikes}} {C_i}^{k+i-1}$. Hence the total number of fault-scenarios for exhaustive characterization is ${n^{2k}}_{quant}(\sum_{i=0...n_{spikes}} {C_i}^{k+i-1})$. It is notable that the number of fault-scenarios is not as large as that for full-functional fault-simulation. For $n_{quant} = 3$ and $n_{spikes} = 5$ (as earlier in Section 2.3), the number of scenarios for a two input operation is 1701. In our studies we have found that two-input operations are the most commonly encountered operations in the automotive domain when the system is partitioned at a reasonable granularity (20-30 operations).

The results of characterization are tabulated in a quality lookup table mapping the quality degradation on the inputs, to the quality degradation of the outputs. Hence it stores a mapping $LUT : \Pi(i_0) \times \Pi(i_1)... \times \Pi(i_{k-1}) \rightarrow \chi(o_0) \times \chi(o_1)... \times \chi(o_{m-1})$. Here $\Pi(i) \subseteq \Pi_{noise}(i) \times \Pi_{shift}(i) \times \Pi_{spike}(i)$, and $\Pi_{noise}(i), \Pi_{shift}(i), \Pi_{spike}(i)$ are the sets of noise, shift and spike values for signal i which are invoked during characterization. The values $\chi(i) \subset R^+ \times R \times Z^+$, denote the values of output noise (positive real), shift (real) and spikes (natural number) for signal i.

2.4.2 Analysis using LUTs

Given tuples of quality degradations for all input signals of an operation op, the quality degradation tuples for the output signals can be approximately estimated from the quality lookup table for the operation *Op*. We use the notation $V(i_l) = <{v_{noise}}^{in}(i_l), {v_{shift}}^{in}(i_l), {v_{spikes}}^{in}(i_l)>$, to denote the quality degradation tuple for input i_l, and $V(o_j) = <{v_{noise}}^{out}(o_j), {v_{shift}}^{out}(o_j), {v_{spikes}}^{out}(o_j)>$, to denote the tuple of quality degradation for output o_j. If the input degradations do not coincide with quality values from characterization, i.e. $V(i_0) \times V(i_1) ... \times V(i_{k-1}) \notin \Pi(i_0) \times \Pi(i_1) ... \times \Pi(i_{k-1})$, then one of several interpolation/extrapolation techniques may be used to estimate the output quality degradations. In this case, the output quality degradations are approximations of the actual quality degradation values. It may be noted that the quality lookup table approximately captures the propagation of quality degradations across an operation, arising from the execution semantics of that operation. Additionally, the operation may itself add to the quality degradation of the signal (due to errors in its own execution), which is manifested as injected faults on the output signals of the operation. Hence, besides the lookup table an 'addition' operation is needed to model the quality response of the operation. This 'lookup table-addition' pair for each operation correspond to the 'operation - fault-injector' pair from the fault-simulation setup described earlier in Section 2.2. There is exactly one lookup table (and addition operations) for each operation in the native model.

It is notable that the lookup table-based analysis is a quality centric analysis. It does not reason about the trajectory of signals, but solely reasons about the quality degradations of signals. Hence it has obvious gains in terms of computational effort needed for analysis. Moreover, the lookup table-based method performs exhaustive local analysis at each operation in the characterization step. This helps the lookup table approach to attain higher coverage for quality-centric analysis.

A lookup table-based estimation of quality degradation is performed by successive table-lookup and addition operations, on a network of lookup tables and addition operations. Fault-scenarios being analyzed are used to add to the quality degradation, just like fault-injection was performed for fault-simulation. For example, consider the partial network of lookup tables and addition operation in Figure 2.7.

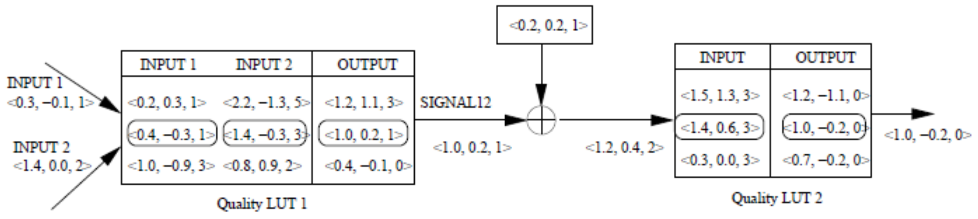

Figure 2.7: Performing Quality-Centric Analysis using LUTs

We illustrate the lookup table analysis for the fault-scenario: $<SIGNAL12, 0.2, 0.2, 1>$, indicating that the signal 'SIGNAL12' undergoes an additional quality degradation of 0.2 units for noise, 0.2 units for shift, and 1 additional spike. The initial quality degradations for signals 'INPUT1' and 'INPUT2 are $<0.3, -0.1, 1>$ and $<1.4, 0.0, 2>$ respectively. This is closest to the lookup table entry '$<0.4, -0.3, 1>$, $<1.4, -0.3, 3>$', as circled in the figure. Hence the propagated value of degradation across the operation is computed to be $<1.0, 0.2, 1>$. Additional quality degradation is added as per the fault-scenario by using the addition operation. This results in the quality degradation at $'SIGNAL-12'$ to be $<1.2, 0.4, 2>$. A lookup in table $'QualityLUT2'$, for another operation OP_2, yields the quality degradation of the output signal of OP_2 to be $<1.0, -0.2, 0>$.

In most designs, like the one presented in Section 2, there is a feedback loop from the outputs of the plant to the inputs of the sensors. This feedback loop is removed for the quality centric analysis, since quality degradations are defined over the complete simulation window (time duration for which simulation is performed) and a lookup table-based analysis without any feedback covers analysis for the simulation window. However, removing the feedback results in the lookup table-based analysis not being able to accurately capture the effects of feedback errors. Additionally, we note that the lookup table-based approach is oblivious to the variations of errors over time, and instead reasons only about the maximum values of errors. In this setup, feedback errors can be modeled by additional sensor errors. However, our experiments indicate that even without feedback error modeling at sensors, the lookup table-based approach tends to overapproximate errors.

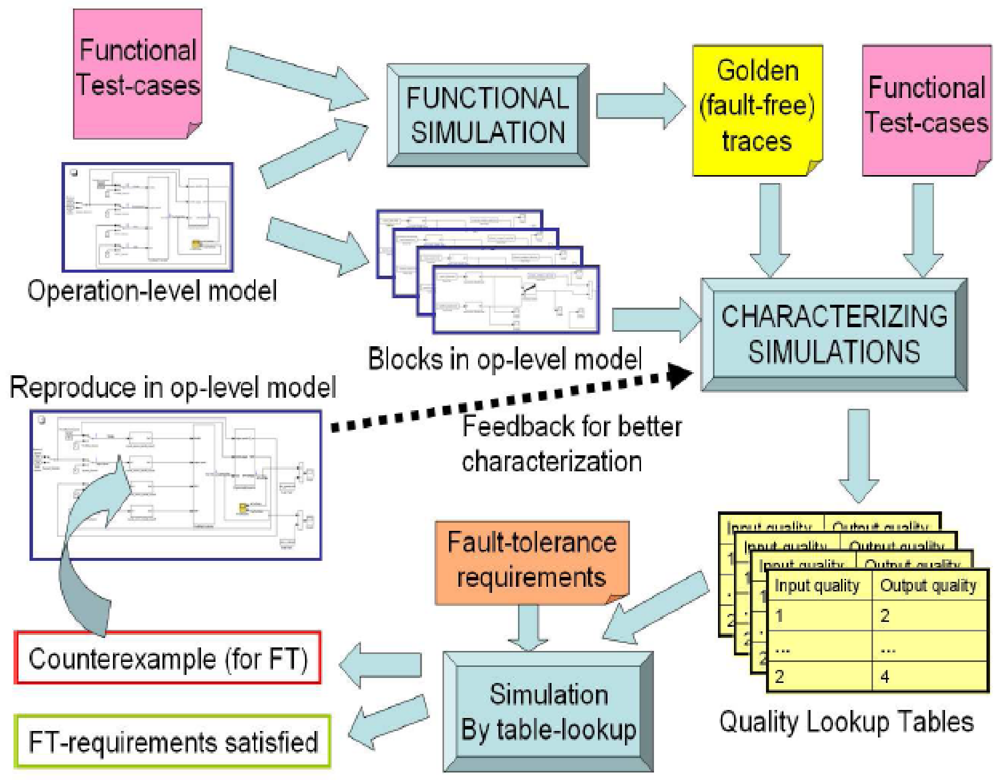
Functional Test-cases
FUNCTIONAL SIMULATION
Golden (fault-free) traces
Functional Test-cases
Operation-level model
Blocks in op-level model
CHARACTERIZING SIMULATIONS
Reproduce in op-level model
Feedback for better characterization
Fault-tolerance requirements
Input quality
Output quality
1
2
...
...
2
4
Quality Lookup Tables
Counterexample (for FT)
Simulation By table-lookup
FT-requirements satisfied

Figure 2.8: Quality Fault Tolerance Analysis Flow

2.5 Quality Fault Tolerance Analysis Flow

In this work we propose an analysis-flow which appropriately melds the lookup table andfault-simulation-based methodologies to leverage both quick simulation and high coverage of the lookup table approach, while maintaining the accuracy of the fault-simulation method. The analysis flow is as follows:

The first step in the analysis-flow is a functional simulation of the operation-level model, in order to obtain golden trajectories at various signals, for each functional testcase. Thereafter, the user partitions the operation-level model into operation blocks, and these blocks are characterized. These characterizing simulations involve fault-injection and simulation of individual blocks, comparing the outputs against stored golden trajectories, and storing signal quality degradation in lookup tables. These lookup tables consist of the input versus output quality of signals for each functional testcase. Once the lookup tables have been obtained, a network of lookup tables and addition operations is constructed, and fault-simulation of the complete system is performed by a series of table lookups. Fault-tolerance requirements are checked on the results obtained by simulation of the lookup table network. If a fault-tolerance requirement is violated, then the counterexample fault-scenario which violates the fault-tolerance requirement is re-examined in the operation-level model of the system by fault-simulation. Hence, the bulk of the fault-tolerance analysis simulations are performed by the lookup table-based approach, while only the counterexample fault-scenarios are examined by the simulation of the operation-level model. Additionally, if there is a significant difference in the values of quality degradations obtained by operation-level fault-simulation and the lookup table-based approaches, then the lookup tables are augmented with quality degradation inputs and outputs at individual operation blocks, as obtained during the fault-simulation. Here, if $I(i_0)...I(i_{k-1})$ are the quality tuples for the input signals of an operation Op, as obtained during fault-simulation, and $O(o_0)...O(o_{m-1})$ are the quality tuples for the output signals, then a new lookup table entry $I(i_0), I(i_1), ...I(i_{k-1}), O(o_0), O(o_1), ...O(o_{m-1})$ is added for the operation Op.

Chapter 3

Modified Approach to Fault Tolerant Analysis

3.1 Need for Modifications

The existing method has several salient features, among them are the ability to handle quality faults, and performing analysis at the operation-level abstraction. Also, it has sufficient coverage and a good deal of accuracy. However, there are several shortcomings of this approach as well. We try to propose certain modifications in the approach, while still keeping the operation-level simulation method and the lookup table-based analysis method as the basic framework of our methodology.

3.1.1 Existing Approach: Interesting Observations

The existing approach has several adavantages as well as several limitations which are worth analyzing. Some interesting observations are as follows:

1. *Representation of Signals.*

   The existing approach uses a combination of noise, shift and spike to capture the behaviour of residue signals. Ideally, this approach tends to have several difficulties. If the spike limit is quite low, then it is easy to confuse between noise and spikes. Also, if the entire signal is shifted by an amount greater than the spike limit in some contiguous time slots(increasing function), then it would be perceived as multiple spikes in the range, whereas actually it should be considered as a shift fault. Fortunately for the system being analysed, the spike limits are much larger in magnitude than the maximum noise, and almost no signals have the output residue as a monotonically increasing function of time. Whatever residues have such behaviour, they are very small in amplitude and hence can be captured by a simple noise term. Thus, this representation is sufficiently accurate for our analysis.

2. *Granularity.*

   Granularity refers to the user-defined quantization levels for noise, shift and spike faults. This indicates the various magnitudes of noise and shift faults that can be added to the golden input signals. Most of the analysis of the existing approach was performed using a granularity of 3 for noise and shift faults. The basic assumption of having a certain granularity is that any value taken from a range, other than the representative value of that range, is assumed to yield the same output residue as done by the representative value. This, of course, leads to a certain amount of inaccuracy in the analysis of components. Nonetheless, having higher granularity increases the simulation time as well as the space required to store the *LUTs* by a large amount. So, there is a distinct trade-off between simulation time and storage space vs the accuracy of simulation.

3. *Functional Testcase.*

   The functional testcase for a component consists of a set of golden input signals and its corresponding golden output signal based on which the fault injection is performed. The existing approach has only one functional testcase for each component, on which perturbations are added and characterization is performed. Clearly, this may not cover all the possible faulty input scenarios. So, the LUTs generated by using just a single functional testcase per component, are somewhat incomplete and the component is thus not completely characterized. We would be much better off, if we had multiple functional test cases, to form the basis for our characterization.

3.2 Proposed Approach

After careful study of the shortcomings of the existing approach, a modified methodology was proposed for performing Fault Tolerant Analysis of Automotive Systems. The entire analysis was done on the *Fault Tolerant Fuel Controller Automotive demonstration* provided by Simulink.The procedure in brief is as follows:

1. *Characterization of Components.* Each component of the automotive system is characterized using multiple functional test-cases. This ensures that a wider variety of faulty scenarios are considered, leading to more complete LUTs. The granularity of analysis is also changed as deeemed suitable to obtain better quality LUTs.

2. *Alternate Representation of Residue.* Although, the existing classification of signals based on noise, shift and spike faults was quite effective, an alternate representation has been suggested to facilitate comparision between output residues.

3. *Gray Box Approach.* The existing approach analyzed each component as a complete black box, using only the input-output residue pairs to characterize its behaviour. But, it is observed that any knowledge of the actual behaviour of the component, i.e. a white box analysis, can help in characterizing the components better. This combination of white box and black box is called as a gray box for every component. So, we try to generate quality LUTs using the gray box approach and study the analysis performed using such LUTs.

4. *Compaction of LUTs.* The more detailed characterization means that the size of the LUTs generated are quite huge and hence, compaction becomes more important than before. An algorithm has been developed to perform compaction of LUTs: both inside an LUT as well as across various LUTs obtained for different functional testcases for a component.

5. *Analysis Using the LUTs.* Using the improved quality LUTs generated for each component, a system level static as well as dynamic analysis is performed, to obtain the number of counter-examples that violate the fault tolerant requirements for the system.

6. *Other Models.*The proposed methodology is used to analyze other automotive systems such as an 'Anti-Lock Braking System' with a view to finding problems,if any, that are inherent to the analysis.

3.3 Characterization of Components

In order to perform any fault tolerant analysis on an automotive system, it is of utmost importance that each component should be characterized as accurately and as completely as possible. Characterization proceeds exactly like fault-injection and simulation as in the previous approach. The fault-scenarios for characterization are generated by adding noise, shift and spike faults on the golden input signals. Even though several types of errors are not injected on many input signals during full-system fault-simulation, characterization requires that all types of errors be injected on each input signal. This is because errors injected at input signals must not only model errors introduced by the immediate source operations, but also model propagated errors which originate beyond the source of the signal, from operations which could introduce any type of error (like sensors). The following steps were followed to characterize the components of the system at hand.

1. *Multiple Functional Testcases.* For each component, we used 5 functional testcases to serve as the golden input-output sets. These were obtained from the golden inputs to these components used in other Simulink models. If those were not available, then random functional testcases(with random spikes and noise) were generated. It was ensured that all the functional testcases had wide variation, so as to cover as much behaviour of the component as possible.

2. *Axes Values and Other Thresholds.* Previously, since only one functional testcase was used, the representative values for the noise and shift axes, and the spike limits, were set by observing the behaviour of that single golden input. In the modified approach, we use the global maximum and minimum among all the signals to set the noise and shift axes values, and to fix the spike upper and lower limits. This ensures that the magnitude of fault added for a given set of noise, shift and spike values is the same irrespective of which functional testcase the perturbation is added on. This makes the input and output residues amenable to comparision.

3. *Modifying Granularity.* Most of the analysis is performed using a granularity of 3, but we do modify the granularity to assist in distinguishing between components. For a component that shows similar output residues to within a specified error limit, for same input residue added across different functional testcases, we do not tinker with the graularity and simply use the LUTs generated at a granularity of 3. But, if a component shows different behaviour, for the same input residue, added to different functional testcases, then we look into changing the granularity to distinguish the components further.

   In such a case, we first reduce the granularity from 3 to 2 and generate the LUTs. If a component still shows different behaviour for different functional testcases, then we cannot combine them in any way, and so, we use the LUTs generated with granularity 3, as they are more accurate. However, if any component generates almost similar LUTs at this reduced level of granularity, then we increase the granularity from 2 to 4 and generate the LUTs. This ensures that we capture more detailed behaviour of such components that showed different behaviour to start off, i.e. we obtain more accurate quality LUTs for such components.

Using this technique more complete LUTs were obtained for each component. Obviously, the size of LUTs increased a lot, so suitable compaction techniques were developed.

3.3.1 Alternate Representation of Residue

An alternate representation was developed to perform comparision between the output residues obtained from different functional testcases. The new chosen representation did not involve classifying the signal in terms of noise,shift and spike, but to store the nature of the signal(its approximate trajectory) using only a small number of data points. Basically, we tried to capture the behaviour of the signal by sampling at specific points in various time slots. We could perform the comparision better by storing the entire signal, but due to space constraints, we resort to the following algorithm.

Algorithm.

1. *Time-slots.* Divide the entire signal into several intervals whose width is user-specified. The width is so chosen that it captures almost the entire signal, keeping in mind the memory available for storing each signal. A width of 100 was found suitable enough as per our memory constraints.

2. *Encoding Scheme.* In each interval, find the maximum and minimum amplitudes of the signal. Store them as (Max-value, Min-value) or (Min-value, Max-value) pairs depending on their order of occurrence in each interval. So, for every interval we store two signal points. For the entire signal store the first and last data points just for the sake of simplification.

3. *Decoding Scheme.* Reconstruct the original signal by plotting these pairs of values at quarter and three-quarters of the interval, in the order in which they were stored. Plot the first and last data points of the encoded signal at their original positions.

In Matlab, each signal is stored using 3001 data points. Using an interval width of 100, we have 30 intervals. For each interval, we store 2 data points, making a total of 60 data points. Taking into account the first and last points, we have encoded a signal using 62 data points. For decoding the signal, it was found that plotting the points at a distance of half the interval length provided the most accurate representation. Since, we no longer represented the signal in terms of noise, shift and spikes, there was no chance of wrong interpretation during comparision.

K-L Divergence.

The Kullback-Leibler divergence, more popularly known as the K-L Divergence is a non-symmetric measure of the difference between two distributions P and Q. This was used to analyze how much different,the output residues of two different functional testcases, were, for the same set of noise,shift and spike input faults. The discrete version of the Divergence is given by : $D_{KL}(P,Q) = \sum_i P(i) \log \dfrac{P(i)}{Q(i)}$.

This usually is used to compare two probability distributions. So, to apply this to any ordinary signal, an appropriate normalization constant was used. The normalization constant for each signal was the sum of the amplitudes at the encoding points of the alternate representation. Using those 62 points for each signal, K-L Divergence was computed for 5 randomly chosen pairs of output residues. These were averaged out to give the K-L Divergence value for that component. These K-L Divergence values provided a technique for categorizing the various components of the automotive system.

3.4 Compaction of LUTs

The LUTs as seen from the calculations above can contain large number of entries if the component is susceptible to all the three types of quality faults. Storing such large LUTs may be a problem. So, one of our aims is to compact the LUTs. The compaction of LUTs can be performed inside every LUT individually, or across LUTs for different functional testcases for a component. While compacting, we basically look for entries that can be merged by virtue of their similar nature, yielding a smaller LUT.

Across a single functional testcase. For each functional testcase, we add perturbations in the form of noise, shift and spike faults and create various faulty input signals (say m in number). Now for any pair of the m input signals, if the output residues are same within a specified error limit, then we can combine the rows for these two signals to obtain a single row in the LUT. For performing such compaction, we use a technique very similar to circuit packing methods, in which we try to form 'solid groups' comprising blocks of LUT portions that have similar values. The amount of similarity, needed for being a member of a solid group, depends on the specified error limit δ. For those solid groups, we combine the rows of LUTs, to generate a range of noise, shift or spike axes values that have almost similar LUT entries. Thus, we can compact across a single functional testcase. These compacted LUTs are in the format necessary to perform static analysis and can improve its efficieny to a large extent.

Across multiple functional testcases. For each of the n functional testcases, we obtain m signals as before. Choose pairs of signals across the testcases. Depending on how much fault tolerant the system is, some or most of such pairs will have similar output residues within a certain error limit. For components that generate almost similar output residues to within a specified error limit, for same input residue added across different functional testcases, we average out the LUT entries for all such pairs to obtain only a single quality LUT for such components. Thus, we have compacted across functional testcases.

The need for compaction increases in the proposed approach, as for certain components, we generate the LUTs using a higher granularity of 4. This means that the tables will be of larger size. Also, we are characterizing the components based on multiple functional testcases, meaning that we will have more number of LUTs for each component. Hence, compaction of LUTs, both inside an individual LUT and across LUTs for a component is very much an essential step in our analysis.

3.5 Static Analysis

It has been observed that logical faults can be effectively analyzed using logic-reasoning methods, and methods like fault-tree analysis. However, the same is not true for quality faults. The analysis of quality faults is usually done by fault-injection and simulation, which is incapable of providing sufficient coverage. As an example, consider a system which has 20 components, each of which can introduce some bounded quality degradation. In this case, even if we were to study a simplistic quality-fault model where a quality fault with a certain amplitude may either occur or not occur, we have to test about one million testcases. The alternative to testing a very large number of testcases is to formulate and solve the quality fault-tolerance problem as a static analysis, or an optimization problem as done in some robustness analysis methods.

In this project, we investigated a static analysis method for performing quality fault-tolerance analysis. The advantage of static analysis over simulation, is that it can store and use partial simulation-run results to build-up complete simulation-runs, without having to redo simulations from the start. Moreover, if any partial-result of one simulation run matches with that of a previously investigated run, then this run is not explored any further (state matching). Additionally, recent developments in static analysis techniques, notably SAT and SMT (Satisfiability Modulo Theory) solving, allows the use of effective search heuristics to address large problems. However, the applicability of static methods is usually limited by available computer RAM memory. In particular, the memory requirements for analyzing operation-level models, like C-code or Simulink/Stateflow models, is such that scalability to handle large programs remains a challenge. Such being the case, it is important to appropriately abstract the software control system behavior, so as to facilitate the use of static analysis, while retaining traceability between results identified by this analysis and the actual system behavior.

We propose a static-analysis-based framework for quality fault-tolerance analysis of operation-level models like Matlab/Simulink. Due to the trend towards designing automotive software in operation-level languages like Matlab/Simulink, and with the intention of identifying issues early in the design-flow, it is important to perform this analysis at the operation-level. The inputs to the analysis framework are: a functional specification of the fault-hypothesis and quality fault-tolerance requirements, in a language proposed in this work; the set of functional testcases; and the operation-level model of the application. The following algorithm is used to perform static analysis for automotive systems with quality faults.

Algorithm.

1. Partitioning the complete operation-level system model into sub-systems components.

2. This is a functional testing step, which creates golden simulation traces for the quality fault-tolerance analysis, for a suite of functional testcases. The input and output traces for each sub-system component, for each functional-testcase, is recorded in this step.

3. The characterization step wherein for each sub-system component, and each functional-testcase, several characterizing simulation runs are performed. Each characterization run consists of perturbing inputs (corresponding to a functional testcase) in a systematic manner and recording the output perturbations. The output of this step is a set of lookup tables (LUTs), one for each sub-system component, and each functional testcase. The characterization step is where the abstraction occurs.

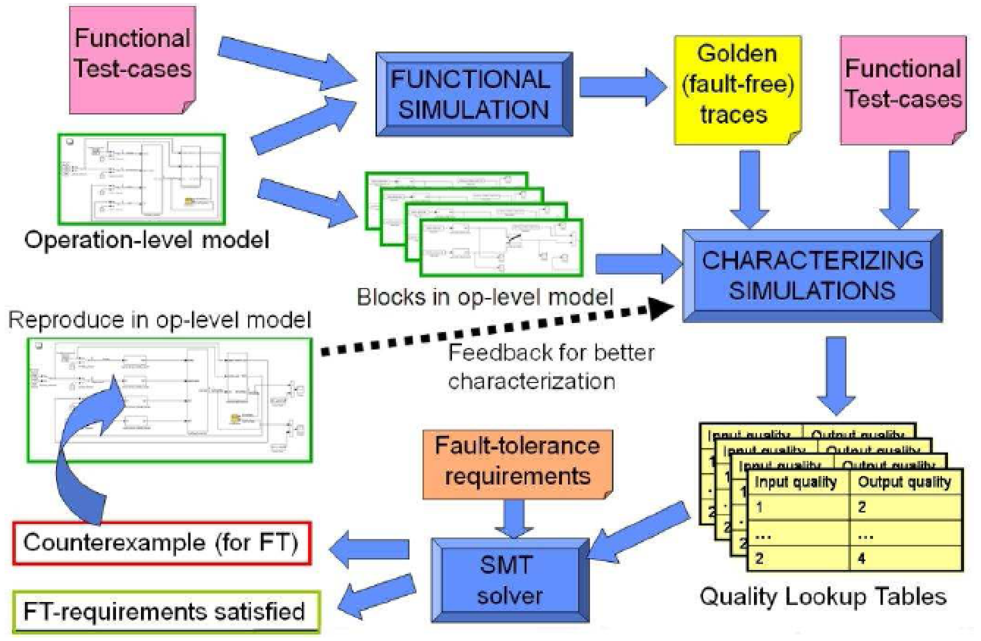

Figure 3.1: Static Analysis : An Overview

Here the detailed dynamics of the system are hidden, and abstracted as quality lookup tables mapping input quality degradation to output quality degradation of individual components.

4. A SMT instance is created by replacing each sub-system operation by a LUT and an addition operation (to model propagation of errors, and introduction of errors by the component respectively), and adding constraints to bind input and output values of sub-system components appropriately. The variables modeling injected faults, and those modeling the error at the outputs are constrained in accordance with the fault-hypothesis and fault-tolerance requirements respectively.

5. Thereafter we use a SMT solver to obtain a fault-scenario which violates the fault-tolerance requirements. This counterexample fault-scenario indicates a hot-spot in the design (abstraction in the characterization step implies that the results of static analysis are approximate).

6. Finally, this counterexample fault-scenario is reproduced in the operation-level model, and checked if it is spurious or an actual fault-scenario.

Spurious counterexamples may also be used for improving the accuracy of characterization.

It is notable that one may also consider the aforesaid approach to be a directed fault-simulation, where the results of the SMT solver provide directed testcases to be simulated in the operation-level model.

3.6 Dynamic Analysis

Any analysis that is simulation-based comes under the category of dynamic analysis. These simulation-based methods are employed on operation-level, as well as implementation-level models for logical and quality analysis. Typical operation-level models used are Simulink models [8], while C/C++ processor simulators, and emulation setups executing automotive software, are employed for implementation-level analysis.

The entire fault simulation based method used in the existing approach is based on dynamic analysis. In a simple sense, dynamic analysis means analyzing a system using LUTs. Basically, we construct the system as a network of LUTs, in which components are connected to each other. We perform simulation on the entire system using the LUT values and analyze the results. The dynamic analyis gives us an idea about the number of counterexamples produced in a set of simulation runs at the operation level, that violate the given fault tolerance requirements. These counterexamples are then run at the system level to verify if they actually violate the system requirements.

Chapter 4

Work Done

The entire fault tolerant analysis was done on the 'Fault-tolerant fuel controller' (ft_fuelsys model) automotive system from the Simulink automotive demonstrations. First, a brief overview of the selected automotive model is presented, followed by the observed experimental results.

4.1 Fault Tolerant Fuel Controller

The 'Fault-tolerant fuel controller' model is an engine-block application, which computes the fuel flow rate for the engine, dependent on various inputs. The Simulink model of the 'Fault-tolerant fuel controller' example consists of clearly demarcated control (Fuel-Rate-Controller) and plant (Engine-Gas-Dynamics) sub-systems. There are two external inputs to the model, namely the throttle and the engine speed. Additionally, the controller has inputs which indicate the oxygen flow (EGO) and the manifold pressure (MAP), connected via a feedback loop from the plant. All controller inputs are obtained from appropriate sensors. The studied output of the system (the monitored signal) is the fuel-flow rate calculated by the control.

We perform an analysis to study the effects of sensor and software component failures on the output (fuel-flow rate) of the control. Sensors suffer from noise, shift, and spike errors, while software components suffer shift and spike errors (only spike errors in case of boolean output), and hardware components suffer from spike errors. A spike error in the sensor is treated as a temporary logical error by the system and the fault-resilience framework

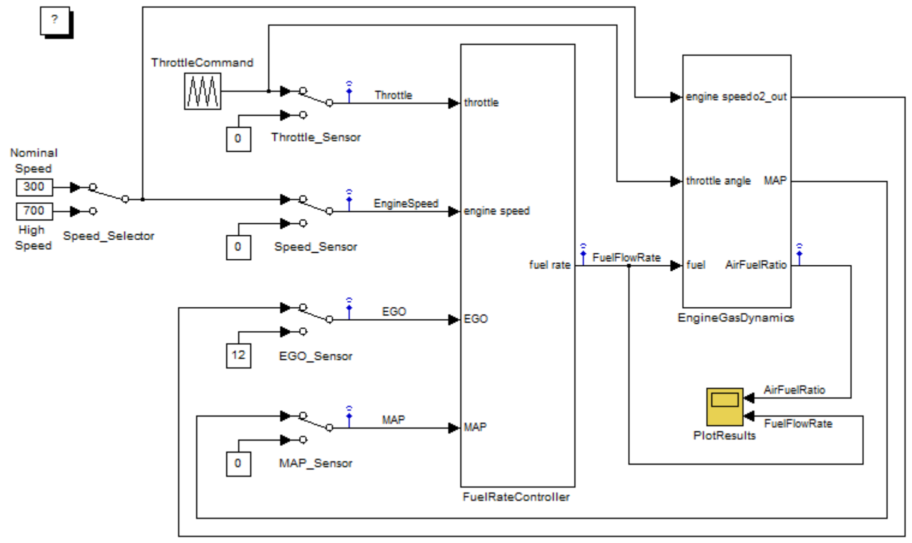

Figure 4.1: A Schematic Operation-Level Model of "The Fault Tolerant Fuel Controller"

(the 'sensor fault correction' block in Figure 4.2) inbuilt into the system tries to correct the errors. However, the recovery usually leads to shift errors on outputs, in lieu of spike errors.

The Simulink model of the ft_fuelsys system consists of two main components, namely the controller (Fuel-Rate- Controller) and the plant (Engine-Gas-Dynamics). The control, consists of four main components, as shown in Figure 4.2. These are the control logic, the sensor fault correction unit, the air-flow calculation unit, and the fuel calculation unit. The Fuel-Rate-Controller is assumed to be implemented in software, and a fixed-step solver (with a time-step of 0.01 s) is employed for simulations. We identify 20 operations in the control component (Fuel-Rate- Controller) of the ft_fuelsys system. Among these components, 15 have two floating-point inputs, 4 have one floating-point and one boolean input, while 1 has one-floating point input.

The fault-tolerance requirement studied for this automotive system is:
Antecedent:'The noise injected at sensor outputs is less than 10 of the maximum value of thesignaltrajectory, and the shift injected at sensor outputs is

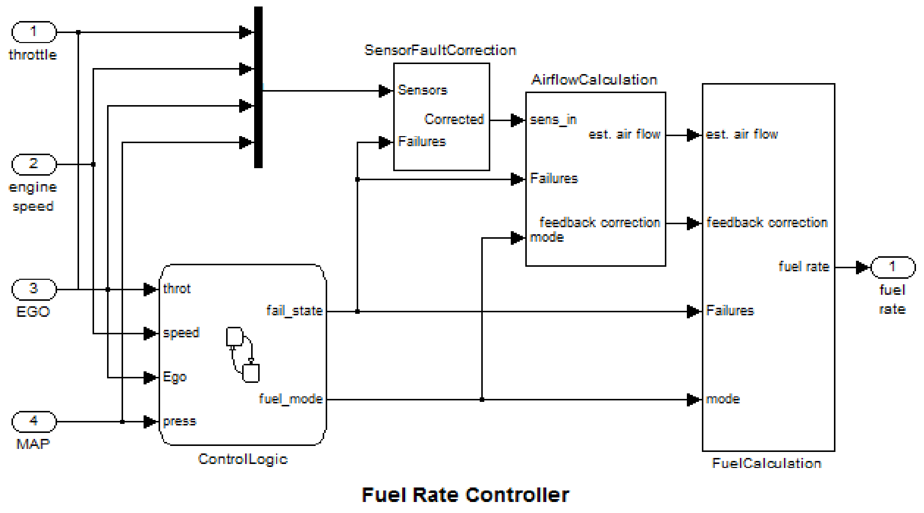

Figure 4.2: The actual controller module

between -5 to 5 of the maximum value of the signal trajectory, and the shift injected at software component outputs is less than 1 of the maximum value of the signal trajectory, and the total number of spikes injected is less than or equal to 5.'
Consequent:'The noise and shift at the fuel-flow rate signal is less than 5 of maximum value of the signal trajectory.'

A random fault-scenario generator for generating fault-scenarios consistent with the antecedent of the fault-tolerance requirements. We utilize these fault-scenarios in the developed fault-simulation framework for quality fault-tolerance analysis. A framework for exhaustive characterization has also been developed. This characterization framework tests individual operations for all fault-scenarios(using multiple functional testcases) at a user defined granularity for shift and noise faults (number of points between 0-10 and -5 to 5 fault-levels respectively), and up to 5 (maximum number allowed by the antecedent) spikes on the input signals. We develop a network of lookup tables and addition operations for fault-simulation using lookup tables.

4.2 Characterization

4.2.1 Study of LUTs

The components of the fault tolerant fuel controller were characterized using multiple functional testcases as mentioned before. Based on the results of comparision of the output residues, the components could be divided into two broad categories:

Type 1. For these components, the nature of the output residues for similar input residue, added to various functional testcases, was same within a small error limit. So, the LUT entries had almost similar values for perturbations added to different golden input sets. Also, these had low K-L divergence values for various output residues. This means that our set of functional testcases was sufficient to almost completely characterize such components. The adder, multiplier, TAM_MAP, TAM_MAR were the two-input components that showed this type of behavior.

Type 2. For these components, the nature of the output residues for similar input residue, added to various functional testcases, was quite varied. The LUT entries had widely different values for different functional testcases, indicating that the output residue was somehow dependent on the golden input values. A white box approach might give useful insights regarding such components. These components also have high K-L divergence values for various output residues. This means that our set of functional testcases is not sufficient to completely characterize such components. Some examples of such components are MC_o2, MC_AFR, oxygen_correction, pumping_constant, map_ estimate, throttle_estimate,etc.

In order to characterize such components better, we might be tempted to use more number of functional testcases, thereby obtaining more number of LUTs. But, there are several limitations to generating large amount of LUTs. First of all, there is a time constraint. Simulation for multiple functional testcases require a large amount of time; so we cannot go on generating LUTs forever. Secondly, the larger the number of LUTs we obtain, the more difficult it becomes to store and process such LUTs. Greater amount of compaction then becomes a major requirement.

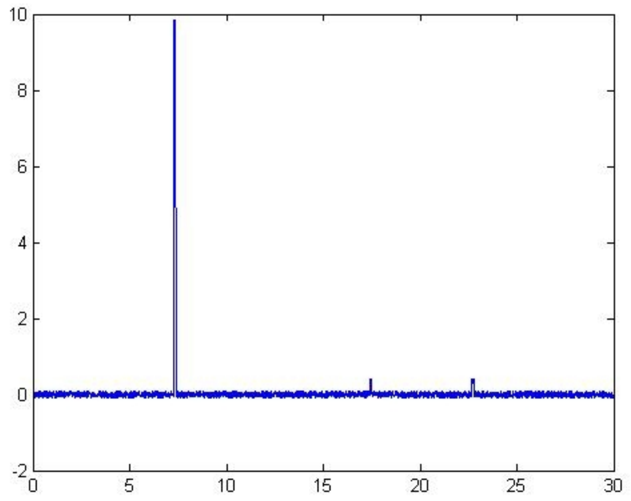

Figure 4.3: Original Signal

4.2.2 K-L Divergence using Alternate Representation

The comparision of two output residues, obtained from same input residue added to different functional testcases, is done using the metric of K-L Divergence. We cannot compare the residues,simply on the basis of LUT values, because sometimes, the spikes may be slightly lower than the specified spike limits and hence may be misinterpreted as noise. This would throw off the LUT values, but the divergence value may still be very small. For obtaining K-L Divergence, we use an alternate representation to capture the basic behaviour of the signal.

Sample signals using Alternate Representation.

K-L Divergence. As the table 4.1 shows, the components of type 1 have a much lower K-L Divergence value than components of type 2. For type 2 components, we can perform a further distinction, based on the faults added to each of the two golden inputs, as follows:

1. *Type 2a.* In such components, both the inputs are susceptible to all three types of faults, viz. noise, shift and spike faults. Examples of

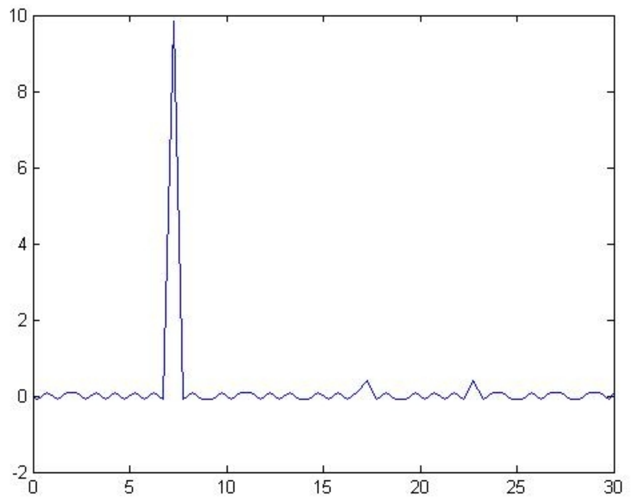

Figure 4.4: Alternate Signal Representation

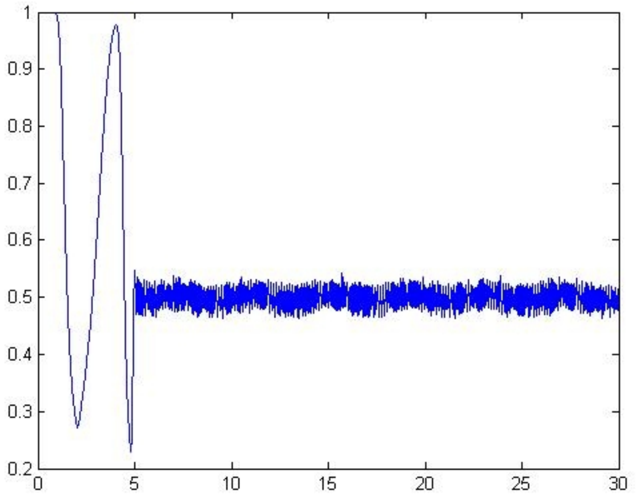

Figure 4.5: Original Signal

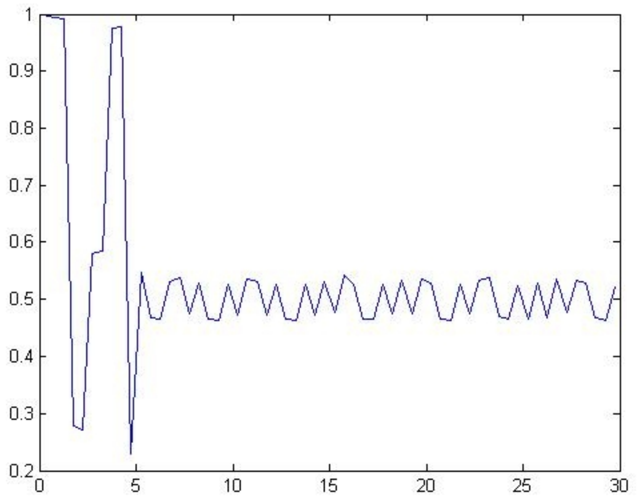

Figure 4.6: Alternate Signal Representation

such components are Oxygen_correction,Pumping_constant, MC_AFR, etc.

2. *Type 2b.* For such components, only one input is susceptible to all three kinds of faults while the other is susceptible to only spike faults. Examples of such components are the integrator module, low_mode, rich_mode and ramp_rate.

It is observed that the Divergence value of Type 2b components is less than the Divergence value of Type 2a components. This is possibly due to the presence of lesser variety of faults in one input for type 2b components. As seen from the table, the K-L Divergence value of the integrator module, a type 2b component, is 23.47 while that of the pumping_constant module, a type 2a component, has a much higher divergence value of 52.51, reinforcing our conclusion.

Faults Other than the Quantized Values. Fault injections were done choosing noise and shift faults other than the representative values inside a single level of granularity. The LUTs generated had somewhat different val-

Table 4.1: K-L Divergence for components

Component	K-L Divergence	Component Type
Adder	0.51	1
Multiplier	0.76	1
TAM_MAP	1.23	1
TAM_MAR	1.45	1
Integrator	23.47	2
Oxygen_correction	32.45	2
MC_o2	37.78	2
MC_AFR	41.38	2
Pumping_constant	52.51	2

ues than that of the ones obtained by using the values of the quantized levels of faults. Nonetheless, the same pattern followed for all type 1 components, i.e. even these generated almost similar output residues for different functional testcases. Hpwever, the type 2 components continued to show wide variation in the output residues, generating different signatures that were dependant on the functional testcase. Again, a white box approach may prove useful to performing such analysis.

4.3 Behaviour of Certain Components

In this section, we present the behaviour of both type 1 and type 2 components, in terms of their LUT values as well as the output residue signals. We choose the simplest type 1 and type 2 components, namely the *adder* and the *oxygen_correction* module respectively.

4.3.1 Adder Module

The adder is a simple component, which has two inputs, both of which are susceptible to all three types of faults, and the output signal is simply the sum of the two input signals.

Consider the following case:

- Granularity = 3 for noise and shift faults and 5 for spike faults

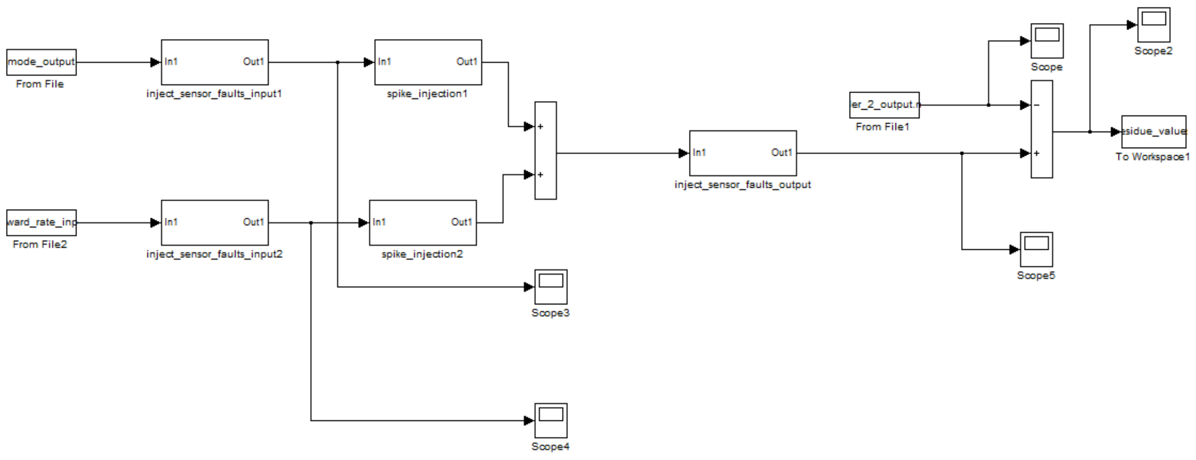

Figure 4.7: The Adder Module

- Input 1: $<noise, shift, spike> = <2, 2, 3>$
- Input 2: $<noise, shift, spike> = <2, 2, 2>$
- Input 1 Axes: noise = [0,367,734], shift = [-367,0,367](scaling factor = 10^{-4} for both shift and noise faults), spike = [0,1,2,3,4,5]
- Input 2 Axes: noise = [0,35,70], shift = [-35,0,35](scaling factor = 10^{-4} for both shift and noise faults), spike = [0,1,2,3,4,5]

Fig. 4.8 shows the injected input residues, while Fig. 4.9 and 4.10 show the behaviour of the output residues for two different functional testcases.

The two output residues are obtained from the same input residue being added to two different functional testcases. As evident from the figures, both these signals are quite similar, with a very low K-L Divergence value of 0.57. The LUT tables corresponding to the above output residues are as follows:

Noise LUTs.

$$LUT_{1,noise}(:,:,2,2,3,2) = \begin{bmatrix} 9483 & 9512 & 9497 \\ 9412 & \mathbf{9565} & 9513 \\ 9383 & 9625 & 9539 \end{bmatrix}$$

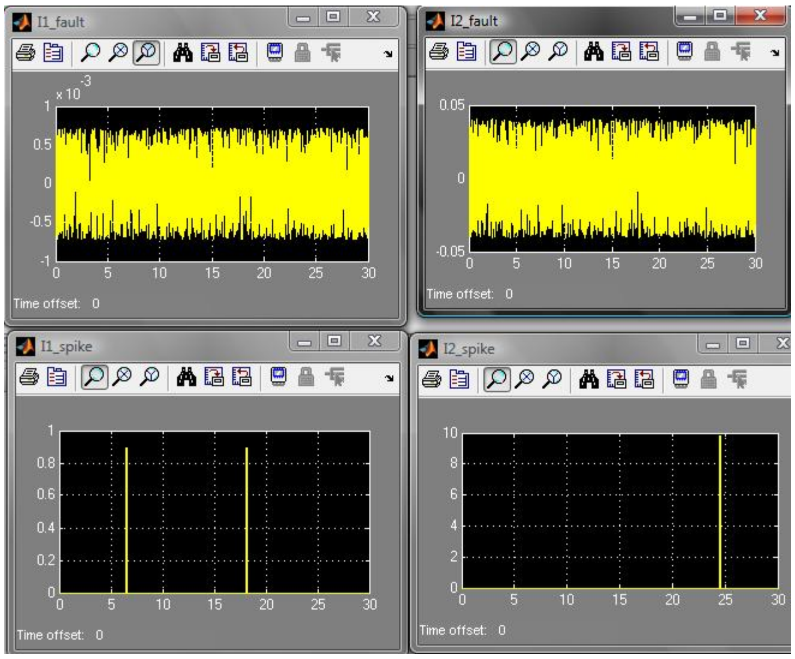

Figure 4.8: Injected Faults into the Golden Inputs

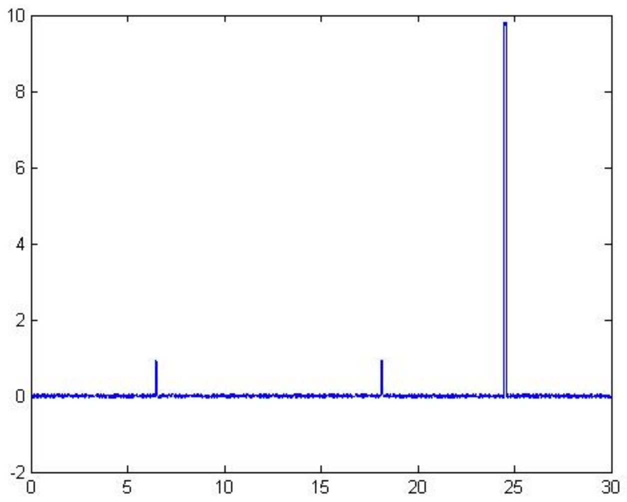

Figure 4.9: Output Residue 1

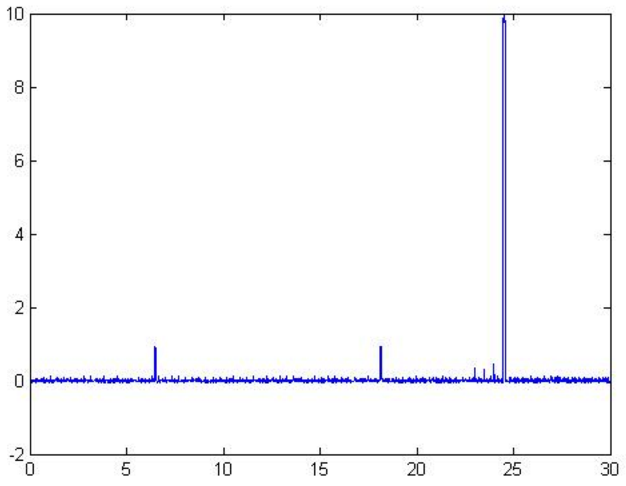

Figure 4.10: Output Residue 2

$$LUT_{2,noise}(:,:,2,2,3,2) = \begin{bmatrix} 9461 & 9533 & 9449 \\ 9407 & \mathbf{9579} & 9487 \\ 9349 & 9658 & 9515 \end{bmatrix}$$

Shift LUTs.

$$LUT_{1,shift}(:,:,2,2,3,2) = \begin{bmatrix} 4867 & 4878 & 4839 \\ 4833 & \mathbf{4895} & 4861 \\ 4821 & 4902 & 4893 \end{bmatrix}$$

$$LUT_{2,shift}(:,:,2,2,3,2) = \begin{bmatrix} 4861 & 4867 & 4844 \\ 4817 & \mathbf{4879} & 4869 \\ 4809 & 4893 & 4881 \end{bmatrix}$$

Spike LUTs.

$$LUT_{1,spike}(:,:,2,2,3,2) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

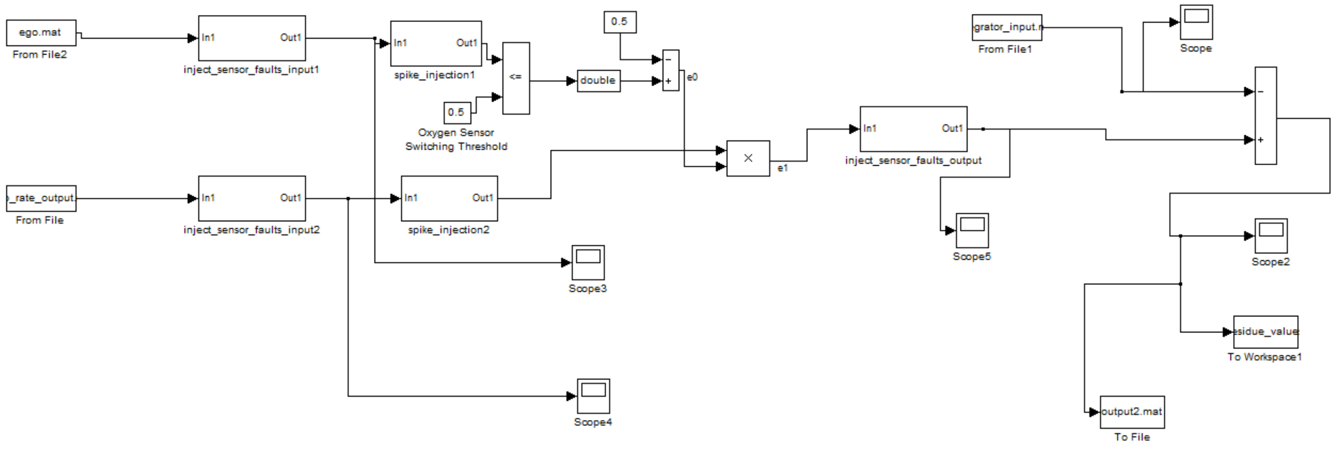

Figure 4.11: The oxygen_correction module

$$LUT_{2,spike}(:,:,2,2,3,2) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Discussion. The numbers in bold indicate the LUT values corresponding to the considered faulty scenario. As one can clearly see, both the signals have near about the same LUT values. This kind of behaviour was observed for all pairs of output residues obtained by using all the 5 functional testcases. Thus, the adder has been completely characterized using our set of testcases.

4.3.2 Oxygen_ Correction Module

The Oxygen_correction module is a simple two input component that controls the amount of oxygen needed for various fuel consumption requirements. Both its inputs are susceptible to all three types of faults, and the output signal represents the oxygen needed for different fuel consumption requirements.

Consider the following case:

- Granularity = 3 for noise and shift faults and 5 for spike faults
- Input 1: $<noise, shift, spike> = <2, 2, 3>$

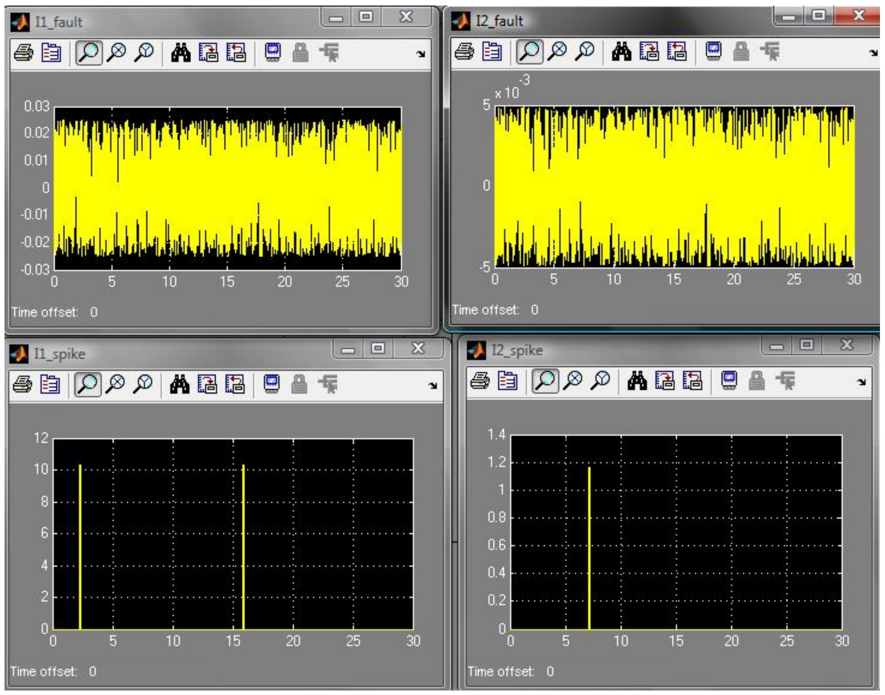

Figure 4.12: Injected Faults into the golden inputs

- Input 2: $<noise, shift, spike> = <2, 2, 2>$
- Input 1 Axes: noise = [0,500,1000], shift = [-500,0,500](scaling factor = 10^{-4} for both shift and noise faults), spike = [0,1,2,3,4,5]
- Input 2 Axes: noise = [0,98,196], shift = [-98,0,98](scaling factor = 10^{-4} for both shift and noise faults), spike = [0,1,2,3,4,5]

Fig. 4.12 shows the injected input residues, while Fig. 4.13 and 4.14 show the behaviour of the output residues for two different functional testcases.

The two output residues are obtained from the same input residue being added to two different functional testcases. As evident from the figures, both these signals are quite different, with a high K-L Divergence value of 33.17. The LUT tables corresponding to the above output residues are as follows:

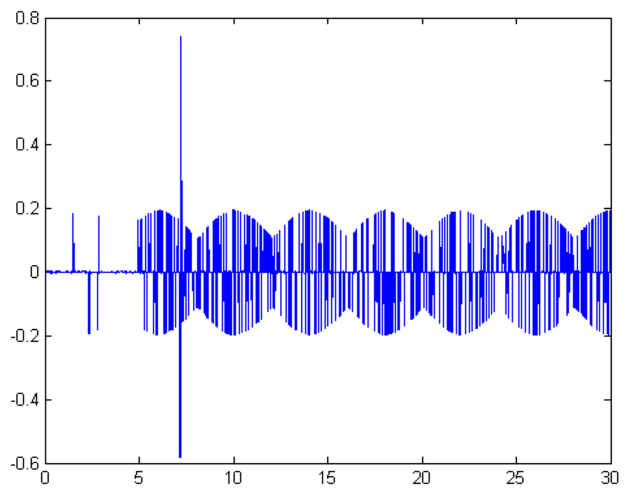

Figure 4.13: Output Residue 1

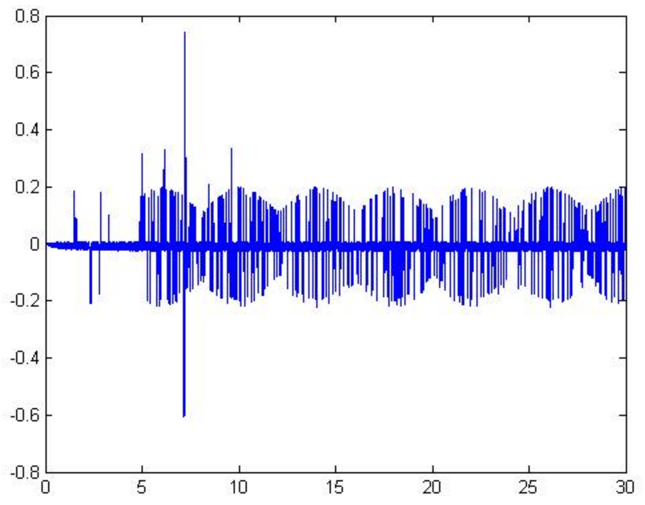

Figure 4.14: Output Residue 2

Noise LUTs.

$$LUT_{1,noise}(:,:,2,2,3,2) = \begin{bmatrix} 7984 & 7872 & 7891 \\ 7921 & \mathbf{7963} & 7912 \\ 7975 & 7944 & 7935 \end{bmatrix}$$

$$LUT_{2,noise}(:,:,2,2,3,2) = \begin{bmatrix} 5716 & 5677 & 5548 \\ 5892 & \mathbf{5783} & 5672 \\ 5613 & 5881 & 5469 \end{bmatrix}$$

Shift LUTs.

$$LUT_{1,shift}(:,:,2,2,3,2) = \begin{bmatrix} -2085 & -2123 & -2129 \\ -2146 & \mathbf{-2135} & -2097 \\ -2131 & -2098 & -2123 \end{bmatrix}$$

$$LUT_{2,shift}(:,:,2,2,3,2) = \begin{bmatrix} 752 & 735 & 729 \\ 861 & \mathbf{745} & 737 \\ 719 & 787 & 827 \end{bmatrix}$$

Spike LUTs.

$$LUT_{1,spike}(:,:,2,2,3,2) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$LUT_{2,spike}(:,:,2,2,3,2) = \begin{bmatrix} 1 & 1 & 1 \\ 2 & \mathbf{2} & 1 \\ 1 & 2 & 1 \end{bmatrix}$$

Discussion. The numbers in bold indicate the LUT values corresponding to the considered faulty scenario. As one can clearly see, both the signals have radically different LUT values. The magnitude of the spike limit for the considered set of functional testcases is set at 0.601. The discernable negative spike has a magnitude of 0.595 in the first residue and a magnitude of 0.607 in the second residue. So, it is not counted as a spike in the first residue but is counted as such in the second residue. Hence, a difference of 1 appears in the spike LUTs. Also, in the first residue, since the negative spike is actually not counted as a spike, it contributes to the noise and shift faults, resulting in higher noise amplitude(LUT values) and a negative shift value. On the other hand, in the second residue, as the negative spike is classified as a spike, the noise is limited only to the observable crests and troughs resulting in lower noise values and the shift also has a positive value. The LUTs of the second residue capture the signal much better than that of the first residue.

This kind of behaviour was observed for all pairs of output residues obtained by using all the 5 functional testcases. Thus, we can clearly see that our characterization is by no means complete, and the LUTs cannot be combined in any way. So, for type 2 components, we need more functional testcases or a different technique for characterization.

4.4 Anti Lock Braking System

An anti-lock braking system, or ABS is a safety system which prevents the wheels on a motor vehicle from locking up (or ceasing to rotate) while braking. It simulates the dynamic behavior of a vehicle under hard braking conditions. The model represents a single wheel, which may be replicated a number of times to create a model for a multi-wheel vehicle. The wheel rotates with an initial angular speed that corresponds to the vehicle speed before the brakes are applied. We used separate integrators to compute wheel angular speed and vehicle speed. We use two speeds to calculate slip, which is determined by:

$\omega_v = \frac{V_v}{R_r}$

$slip = 1 - \frac{\omega_w}{\omega_v}$

where:

- ω_v = vehicle speed divided by wheel radius
- V_v = vehicle linear velocity
- R_r = wheel radius
- ω_w = wheel angular velocity

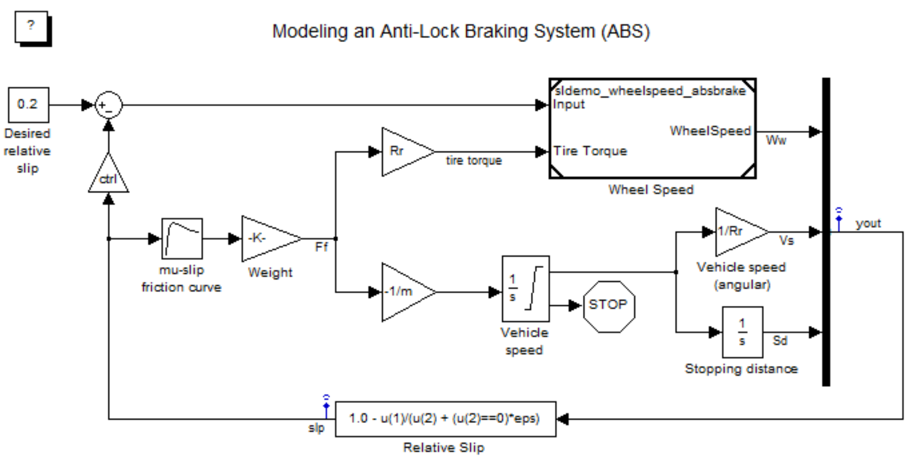

Figure 4.15: The Anti Lock Braking System

From these expressions, we see that slip is zero when wheel speed and vehicle speed are equal, and slip equals one when the wheel is locked. A desirable slip value is 0.2, which means that the number of wheel revolutions equals 0.8 times the number of revolutions under non-braking conditions with the same vehicle velocity. This maximizes the adhesion between the tire and road and minimizes the stopping distance with the available friction. In this model, we used an ideal anti-lock braking controller, that uses 'bang-bang' control based upon the error between actual slip and desired slip. We set the desired slip to the value of slip at which the mu-slip curve reaches a peak value, this being the optimum value for minimum braking distance.

4.4.1 Slip_fcn_Mux unit

This module is actually a combination of a MUX and a simple mathematical module that computes the relative slip. It behaves exactly like a type 1 component of the previous system. In fact, except the integrator module, all other components of the ABS are of type 1, i.e. they have similar output residues for same input residues, irrespective of the golden input set.

Consider the following case:

- Granularity = 3 for noise and shift faults and 5 for spike faults
- Input 1: $<noise, shift, spike> = <2, 2, 3>$

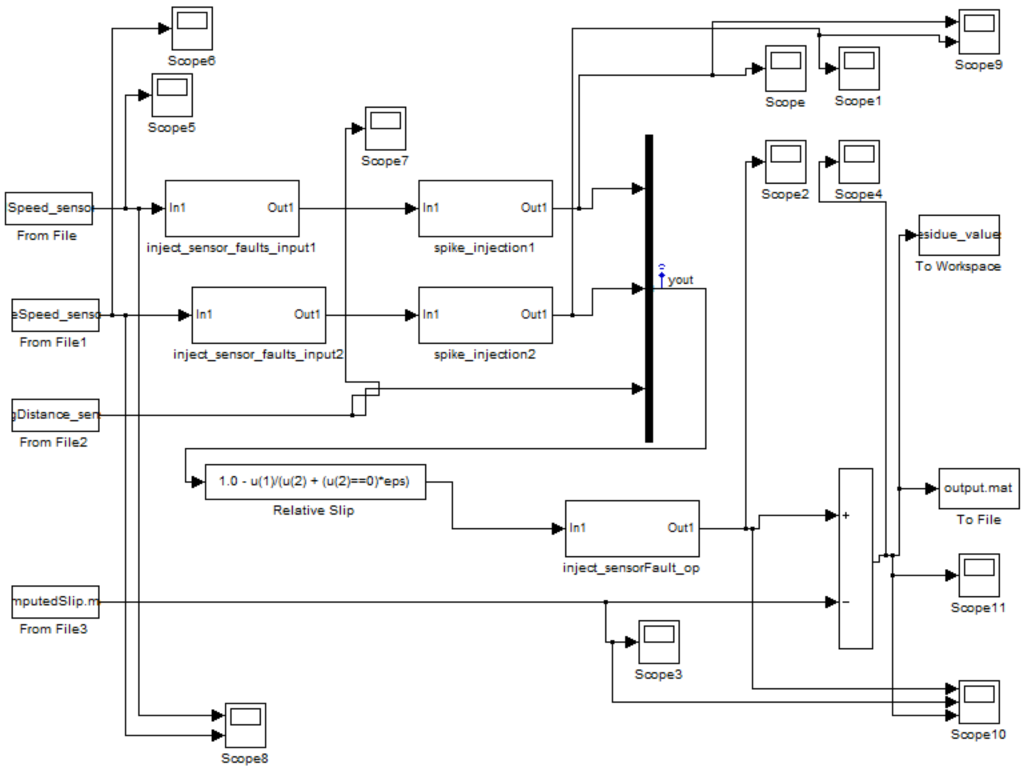

Figure 4.16: The slip_fcn_mux model

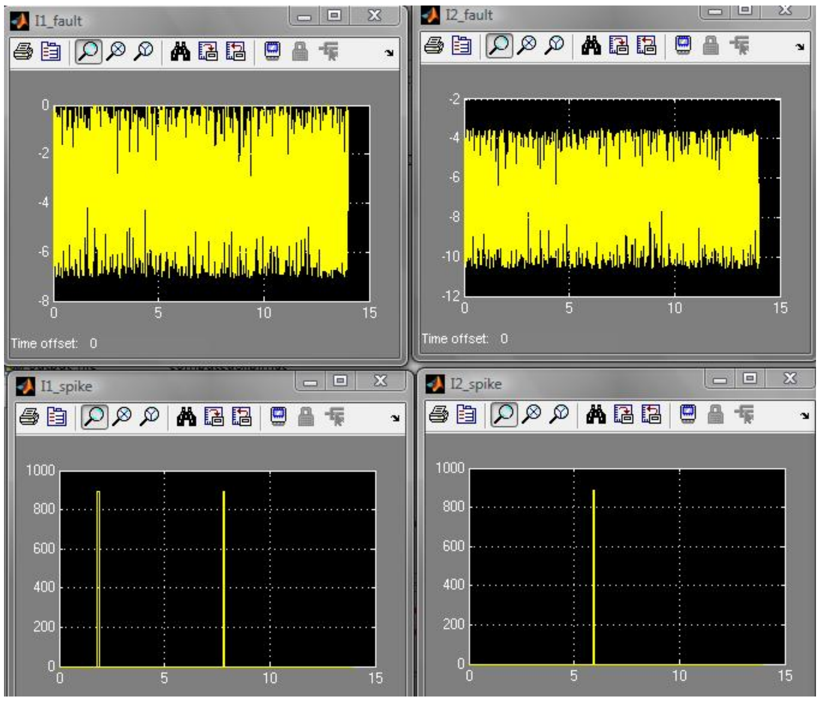

Figure 4.17: Injected Faults into the golden inputs

- Input 2: $<noise, shift, spike> = <2, 2, 2>$
- Input 1 Axes: noise = [0,351943,703885], shift = [-351943,0,351943](scaling factor = 10^{-4} for both shift and noise faults), spike = [0,1,2,3,4,5]
- Input 2 Axes: noise = [0,351999,351999], shift = [-351999,0,351999](scaling factor = 10^{-4} for both shift and noise faults), spike = [0,1,2,3,4,5]

The two output residues are obtained from the same input residue being added to two different functional testcases. As evident from the figures, both these signals are quite similar, with a very low K-L Divergence value of 1.24. The LUT tables corresponding to the above output residues are as follows:

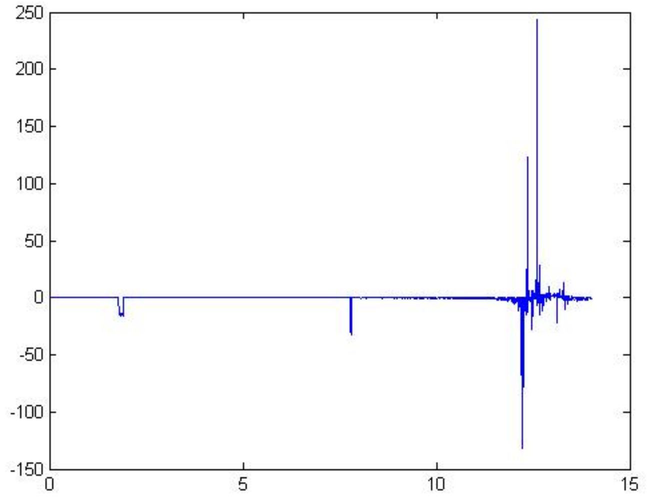

Figure 4.18: Output Residue 1

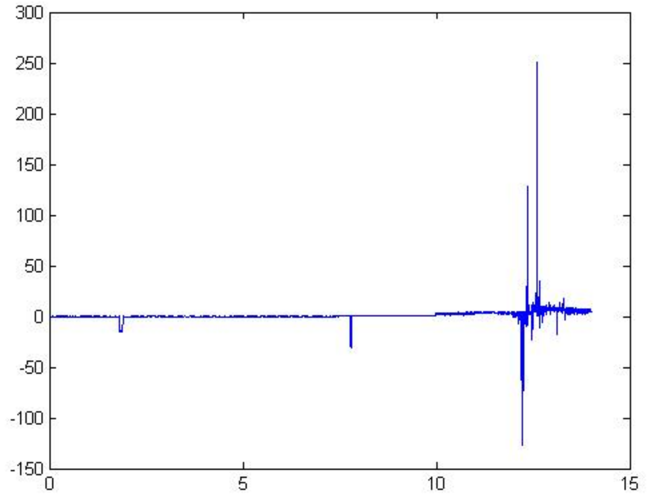

Figure 4.19: Output Residue 2

Noise LUTs.

$$LUT_{1,noise}(:,:,2,2,3,2) = \begin{bmatrix} 680567 & 681134 & 682119 \\ 679154 & \mathbf{678715} & 679226 \\ 680013 & 679718 & 682043 \end{bmatrix}$$

$$LUT_{2,noise}(:,:,2,2,3,2) = \begin{bmatrix} 698436 & 684751 & 687822 \\ 683155 & \mathbf{680561} & 681138 \\ 689342 & 680145 & 682249 \end{bmatrix}$$

Shift LUTs.

$$LUT_{1,shift}(:,:,2,2,3,2) = \begin{bmatrix} 64267 & 64378 & 63839 \\ 63833 & \mathbf{64195} & 64361 \\ 63921 & 64102 & 63893 \end{bmatrix}$$

$$LUT_{2,shift}(:,:,2,2,3,2) = \begin{bmatrix} 64161 & 64267 & 64044 \\ 63817 & \mathbf{64179} & 64809 \\ 64209 & 64393 & 64081 \end{bmatrix}$$

Spike LUTs.

$$LUT_{1,spike}(:,:,2,2,3,2) = \begin{bmatrix} 3 & 3 & 3 \\ 3 & \mathbf{3} & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

$$LUT_{2,spike}(:,:,2,2,3,2) = \begin{bmatrix} 3 & 3 & 3 \\ 3 & \mathbf{3} & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

Discussion. The numbers in bold indicate the LUT values corresponding to the considered faulty scenario. As one can clearly see, both the signals have near about the same LUT values. This kind of behaviour was observed for all pairs of output residues obtained by using all the 5 functional testcases. Thus, the slip_fcn_mux has been completely characterized using our set of testcases.

4.5 Dynamic Analysis

Using the modified characterized data, i.e. the new axes values and new LUTs for each component, the dynamic analysis was performed and compared with the existing approach. The results are tabulated as follows:

Table 4.2: Dynamic Analysis: Using old and new LUTs

Fuel Fault Rate(%)	Counterexamples generated using old LUTs (No. of Potential Candidates)	Counterexamples generated using new LUTs (No. of Potential Candidates)
5	1164(1171)	1143(1147)
6	1127(1171)	1124(1147)
7	1053(1171)	1089(1147)
8	972(1171)	1014(1147)
9	887(1171)	944(1147)
10	799(1171)	872(1147)

The dynamic analysis was performed on a high speed processor for one hour in both cases. The algorithm generates the number of cases that violate fault barriers starting from 5% to 10%. The numbers in the bracket indicate the no. of faulty cases that the dynamic analysis algorithm expects to generate. The other numbers indicate how many counterexamples(cases that violate the fault tolerance requirements) were actually produced. It can be clearly seen that the percentage of counterexamples generated using the modified LUTs is more than that generated using the exisiting LUTs. This indicates that the modified characterization technique is actually effective and generates better quality LUTs.

Chapter 5

Conclusions and Action Plan

5.1 Conclusions

The existing method for fault-tolerance analysis of automotive systems has several salient features, including the ability to handle quality faults, and performing analysis at the operation-level abstraction. This method can identify certain types of accumulated loss of precision (or quality) in automotive systems, which may lead to failures. Additionally, it can abstract the manifestation of automotive system failures to the operation-level, thereby enabling all analysis steps to be performed on operation-level models like Simulink. This facilitates quicker analysis, as well as system simulation for longer time durations wherein quality errors accumulating over time may be unearthed.

Two methods work in tandem for providing fault-tolerance analysis, namely an operation-level simulation method and a lookup table-based analysis method. The operation-level simulation method is based on fault-injection and simulation-trace analysis. On the other hand, the lookup table analysis method quickly analyzes a larger number of fault scenarios by ignoring details of operation behavior and concentrating on quality analysis. This method is based on characterizing individual sub-systems, and storing the variation in output quality versus input quality as a lookup table. Through experiments on an automotive case study, the efficacy of using lookup tables to abstract the quality response of operations was studied. It was found that the number of simulation runs needed for exhaustive fault-analysis is very large; the lookup table-based method helps in providing a quick, but approximate answer, and significantly higher coverage. However, the completeness and accuracy of characterization play a major role in the usefulness of quality LUTs.

We retain the same basic approach of using the lookup table-based and fault-simulation-based methods in tandem, as it ensures good coverage, while maintaining the accuracy of analysis. However, keeping the importance of LUTs in mind, in order to improve their accuracy, a new technique for characterizing components was developed. It characterized certain components very accurately and almost completely while others had to compromise with only slightly improved versions of LUTs. Nonetheless, the improved LUTs were used to perform dynamic analysis on the system. The new LUTs provided very encouraging results with the number of counterexamples generated about 6% more than that of the old LUTs. However, their is still scope for improvement in the dynamic analysis technique. Currently, we have not yet examined whether the counterexamples produced are actually very close to each other or far apart. Ideally, we would like them to be as distinct as possible to cover as much faulty cases as possible.

5.2 Action Plan

The modified techniques have certainly been a step closer towards getting a generic, robust method for performing fault tolerant analysis of automotive systems. But the scope for further analysis is huge. We take a look at some of the steps that we would like to follow in the near future.

Static Analysis. Exhaustive analysis of quality fault-tolerance is a computationally intensive problem. A method has been proposed which suitably abstracts the behavior of operation-level models, to discrete lookup tables, and then performs static analysis to identify hotspots in the design. These hotspots are then analyzed in the operation-level model for checking their validity. The proposed static analysis-based flow may also be treated as a directed testing mechanism. A tool-chain has been developed to perform this analysis. As a first step, we would like to use the modified LUTs to perform this static analysis and compare the results with those obtained from using the old LUTs.

Compacted LUTs. Two algorithms for LUT compaction within the LUT have been developed. One of them is a very fast algorithm that provides 25-35% compaction. It is highly interesting to utilize these compacted LUTs to perform static and dynamic analysis, to examine the accuracy of LUTs preserved under the compaction procedure. Meanwhile, other compaction techniques are also being looked into.

Gray Box Approach Analyzing an automotive system component, both in terms of a white box as well as a black box can provide useful insights to understanding the residue behaviour of the component. So, a gray box analysis of the system at hand is to be performed to obtain any additional information about the components, that could prove useful in modifying the LUTs for better static and dynamic analysis.

Fault Injections Currently, the fault injections for dynamic analysis are not very elegant. For, example a single fault value is chosen from a very wide range at random. We would like to quantize these ranges into more useful intervals, in order to ensure that the counterexamples produced are actually very distinct from each other. Also, we would like to limit the number of fault injection points for better handling of the analysis.

Increasing Granularity Given that all the compaction techniques work properly, we can look into increasing the granularity of analysis as it will obviously improve the accuracy of the LUTs. We hope that using the modified characterization technique at a high granularity, we can come up with much improved LUTs for performing fault tolerant analysis.

Bibliography

[1] Rajeev Alur, Aditya Kanade, S. Ramesh, and K. C. Shashidhar. Symbolic analysis for improving simulation coverage of simulink/stateflow models. In *EMSOFT 08: Proceedings of the 8th ACM international conference on Embedded software*, pages 8998, New York, NY, USA, 2008. ACM.

[2] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, Maurizio Peri, and Saverio Pezzini. Fault-tolerant platforms for automotive safety-critical applications. In *CASES 03: Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*, pages 170177, New York, NY, USA, 2003. ACM.

[3] P. Bellini, R. Mattolini, and P. Nesi. *Temporal logics for real-time system specification,ACM Comput. Surv.*, 32(1):1242, 2000.

[4] Ashok Chandy, Mark Phillip Colosky, and Mark Dennis Kushion. *Robust fault detection method. In US Patent 6,389,338 B1*, May 14, 2002.

[5] Krishnendu Chatterjee, Arkadeb Ghosal, Thomas A. Henzinger, Daniel Iercan, Christoph M. Kirsch, Claudio Pinello, and Alberto Sangiovanni-Vincentelli. *Logical reliability of interacting real-time tasks. In DATE 08: Proceedings of the conference on Design, automation and test in Europe*, pages 909914, New York, NY, USA, 2008. ACM.

[6] F. Corno, P. Gabrielli, and S. Tosato. *Relating vehicle-level and network-level reliability through high-level fault injection. In HLDVT 03: Proceedings of the Eighth IEEE International Workshop on High-Level Design Validation and Test Workshop*, page 71, Washington, DC, USA, 2003. IEEE Computer Society.

[7] Dipankar Das, P. P. Chakrabarti, and Rajeev Kumar. *Functional verification of task partitioning for multiprocessor embedded systems. ACM Trans. Des. Autom. Electron. Syst.*, 12(4):44, 2007.

[8] Jon Friedman. *Matlab/simulink for automotive systems design. In DATE 06: Proceedings of the conference on Design, automation and test in Europe*, pages 8788, Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[9] R. Hoseinnezhad and A. Bab-Hadiashar. *Missing data compensation for safety-critical components in a drive-by-wire system.* Vehicular Technology, IEEE Transactions on, 54(4):13041311, July 2005.

[10] Viacheslav Izosimov, Paul Pop, Petru Eles, and Zebo Peng. *Design optimization of time and cost-constrained fault-tolerant distributed embedded systems. In DATE 05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 864869, Washington, DC, USA, 2005. IEEE Computer Society.

[11] Mark John Jordan, Mark Maiolani, and Andreas Both. *Fault-tolerant electronic braking system. In US Patent 6,540,309 B1*, April 1, 2003.

[12] Stefan Kueperkoch, Wolfgang Grimm, Aleksandar Kojic, Jasim Ahmed, Jean-Pierre Hathout, and Christopher Martin. *Fault-tolerant vehicle stability control. In US Patent 7,245,995 B2*, July 17, 2007.

[13] Nancy Lynch, Roberto Segala, and Frits Vaandrager. *Hybrid i/o automata. Inf. Comput.*,185(1):105157, 2003.

[14] M. Mahmoud, J. Jiang, and Y. Zhang. *Stochastic stability of fault tolerant control systems in the presence of noise. American Control Conference, 2000. Proceedings of the 2000*, 6:42944298 vol.6, 2000.

[15] P. Ramachandran, P. Kudva, J. Kellington, J. Schumann, and P. Sanda. *Statistical fault injection. Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference*, pages 122127, June 2008.

[16] Jeonghee Shin, Victor V. Zyuban, Zhigang Hu, Jude A. Rivers, and Pradip Bose. *A framework for architecture-level lifetime reliability modeling. In DSN*, pages 534543, 2007.

[17] Dipankar Das, P. P. Chakrabarti, and Purnendu Sinha. *Operation-level analysis of quality degradation faults using characterized quality behaviors of components*. *GM-IITKGP CRL*, September 2009