Design Verification of Automotive Electrical Circuits

Alex McManus, Chris Price, Neil Snooke, Richard Joseph

{agm, cjp, nns, rij}@aber.ac.uk
Department of Computer Science,
University of Wales, Aberystwyth
United Kingdom

Abstract

Automotive engineers are under increasing pressure to produce correct, safe designs in shorter time frames. We present an algorithm for automating Design Verification of automotive electrical circuits, building on the qualitative simulator used by the AutoSteve system for producing Failure Mode and Effects Analysis (FMEA). Our system allows an engineer to use a specification (in the form of a state chart) to verify the correct behavior of a circuit design. Design Verification can be used to reveal errors in the specification, sneak circuits, timing and sequence errors, and we give early results for some of these problems.

Introduction

Background

Automotive engineers are under increasing pressure to produce correct, safe designs in shorter time frames. These designs must meet stringent quality and safety requirements. Vehicle designs have been increasing in complexity for many years. In the electrical domain, automotive circuits are often based around sophisticated Electronic Control Units (ECUs), and complex features such as Car Area Networks (CANs) are becoming more common. As design complexity has increased, it has become more difficult for designers to comprehend all the possible ramifications of a failure within their design, and to detect all of the possible interactions between parts of the design.

There is a second trend in vehicle design in addition to increasing complexity. It is a trend towards shorter product design life-cycles. This has prompted a thrust towards concurrent engineering, where design activities are carried out concurrently wherever possible in order to reduce development time and understand all of the interactions between different aspects of the design.

In order to overcome these difficulties, engineers are turning to computers for help. The AutoSteve system helps produce Failure Mode and Effects Analysis (FMEA) reports for electrical circuits effectively and efficiently [1, 2, 3]. Further work has automated some Sneak Circuit Analysis (SCA) [4]. Both of these tools use the same underlying qualitative circuit simulator QCAT. Using a qualitative simulator allows the analyses to be performed early in the design process when changes are less costly to make. Qualitative component models also tend to be more reusable than the equivalent models for a quantitative simulator such as SABER or SPICE. QCAT allows engineers to attach functional labels to the output states of circuits, to help generate human-readable results [5].

Significant previous work in design verification was done by Iwasaki and Chandrasekaran [6]. They use a standard functional representation of the type in Sembugamoorthy and Chandrasekaran [7] to describe both the function of a system and how it is to be achieved. They use the description of overall behavior to verify that a qualitative simulation of a system achieves the expected function, and achieves it in the expected way. One problem with this approach is that the method used to describe overall system behavior does not match the kind of descriptions produced by engineers to describe overall system behavior. This means that the models are not easily available in industry. In the automotive industry, engineers are moving towards using techniques such as state charts [8] to produce a state-based specification of the operation of a system.

This paper describes the use of QCAT to verify the correctness of automotive electrical designs under normal (non-failure) conditions. Currently, automotive engineers perform design verification on an informal, ad-hoc basis. This does result in some errors being missed. We hope that an automated tool will help design verification become a formal part of the automotive design process.

Overview

Figure 1 shows an overview of the design verification algorithm. The design verification tool requires that the engineer specifies the required behavior of a subsystem using a state chart. It then generates an attainable envisionment from the subsystem circuit design (schematic), which effectively results in another state chart. The specification state chart is an abstract, high-level description of the subsystem, which may include nested and concurrent states. In contrast, the envisionment is very low-level, flat and without concurrency. This prevents direct comparison of the two state charts.

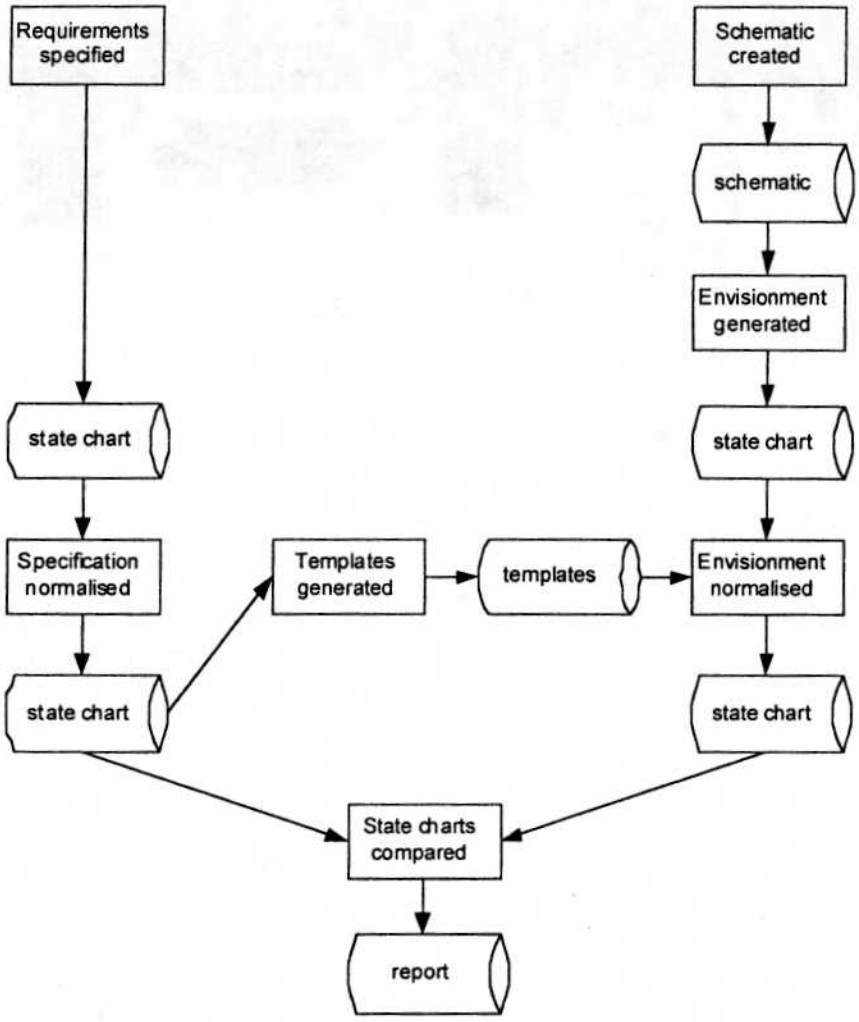

Figure 1: Algorithm Overview

Two levels of normalization are performed so that these state charts can be compared. The specification is normalized to remove concurrent and nested states. It produces templates that are used to guide the normalization of the envisionment, which leads to a new state chart at the same resolution as the specification. A report is generated that describes any differences between the specification and the design.

Section 2 describes how state charts are used to specify the required behavior of a subsystem, and how the state charts are normalized to remove concurrency and nesting. Section 3 details how the envisionment is generated, with section 4 showing how this is mapped to the specification. Section 5 examines how the results are generated, in a form that is meaningful to an engineer. We present some early results in section 6.

Specification

The correct behavior of a subsystem is defined as a state chart (a simple one is shown in figure 1). State charts are expressive enough to capture complex behavior, and are relatively easy for an engineer to define. Engineers using AutoSteve already use state charts to manipulate the low-level component structure, in order to implement complex component behavior in the model definitions.

The state chart notation that we use is a slightly simplified version of that proposed by Harel. Each state specifies which subsystem functions should be active when that state is active. The transitions between states have conditions based on time delays or on the subsystem interface (the inputs to the subsystem, such as switch settings, etc.). The behavior of the subsystem must be expressed in the structure of the state chart, and so local variables are not allowed.

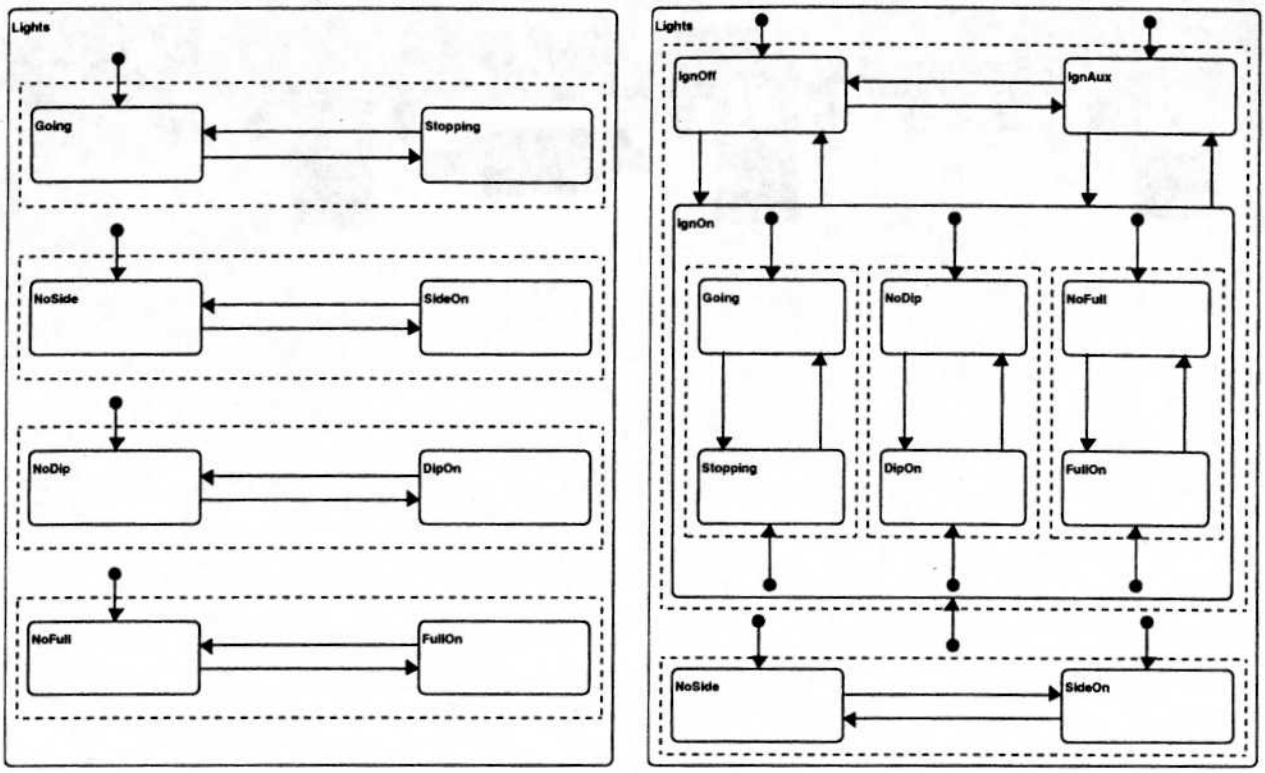

Figure 2: Two Specifications for an Exterior Lighting Subsystem

For a given subsystem, it may be possible to specify its behavior using different state charts. For example, figure 2 shows two specifications for a subsystem, the first one decomposed by subsystem outputs, and the second by subsystem inputs. To get a flat, uniform specification that the envisionment can be mapped to, the state chart is normalized. This involves the following operations:

1. All concurrency is removed. This is a matter of merging all combinations of states and generating the transitions between them. Clearly there is a combinatorial explosion here, but if the behavior were so complex that this became a problem, the attainable envisionment would be computationally infeasible.
2. Nested states are flattened. This involves merging the active functions defined in the parent and child states, and merging/redirecting the transitions connected to them.
3. Unreachable states in the specification are removed (to make it an attainable specification).
4. Redundant states are merged. If two states have the same active functions and the same outgoing transitions (condition and destination state), one of them can be considered redundant.

For a given subsystem behavior, any correct and complete specification will be normalized to the same state chart (subject to the restriction on local variables). A symbol table is maintained during the normalization process, which allows a normalized state or transition to be traced back to the original specification.

Envisionment

The design verification tool generates an attainable envisionment of the subsystem schematic, using the QCAT simulator. The basic algorithm is as follows:

```
initialize subsystemStateList with the
                    initial subsystem state
for each state in subsystemStateList
   for each transition possible from
                              subsystem state
      run QCAT and fire the transition
      if the resulting state is a new
                                    one then
         add it to the subsystemStateList
      end if
   end for
end for
```

From each circuit state, the simulator must try every possible combination of subsystem interface settings (the state of the inputs into the subsystem, such as switch settings etc.). Also, if the circuit features time delays, these can lead to further transitions. The result of the envisionment is effectively a state chart, at a higher resolution than the specification.

Unfortunately, the computational complexity of generating an envisionment is exponential, based on the number of inputs to the subsystem and the number of internal states. In practice however, envisionments of realistically complex subsystems can be generated in a few hours. The envisionment need not be regenerated unless the circuit is changed.

Mapping Envisionment to Specification

At this stage both the normalized specification and the envisionment are flat state charts. The low-resolution envisionment must be normalized to bring it to the same resolution as the specification. In order to use the specification to guide this normalization, a mapping is created between the specification and the envisionment. If the schematic correctly implements the specification, the two state charts will be identical.

Template generation

A template is created for every state in the normalized specification. This consists of all functions that are achieved in that state and the interface settings of the subsystem. The interface settings for a state are calculated by:

1. Getting a combined exit condition for the state, by ORing all exit transitions.
2. Negating this combined condition.

Once the state has become active it will remain so until one of the exit conditions is satisfied – this means that the entry conditions to the state are not relevant.

Merging states

Each envisionment state is compared with the set of templates, and assigned to the template with the closest match. For a template to match, the set of active functions must also be active in the state. The match value will be higher if the state also matches the template's interface settings. It is possible for a state to match more than one template – if this is the case, the connectivity of the state is used to try to resolve the ambiguity. If the implementation of the subsystem is sufficiently different to its specification, it is possible that some of the envisionment states will not match a template.

Each group of states that match to a template are merged to produce a single state that has a direct correspondence to a state in the normalized specification. Every state in the group will have the same set of active functions (in order to match the same template). The conditions of multiple transitions between the same pair of states are ORed together to give a single transition. Transitions between states in the same group are discarded.

Interpreting Results

It is now possible to perform a direct comparison between the normalized specification and the normalized envisionment. Missing transitions, extra transitions and transitions with different conditions are noted. If there are a large number of discrepancies, this information can be difficult for an engineer to interpret. Two approaches to managing this complexity are used: analyzing the entry and exit conditions of each state, and guiding the engineer through a wizard-style user interface.

Because the report discusses the discrepancies in terms of the specification state chart, this makes it easier for the engineer to understand the results.

Comparing the entry conditions to states

For each state in the specification, the transitions entering the state are compared. By comparing the transition's condition in the normalized specification with the condition in the normalized envisionment, it can be determined under which circumstances entry to that state is illegal or restricted (according to the specification). The engineer need not compare the two conditions explicitly.

Grouping these comparisons by state gives them a logical ordering. If more than one transition entering the state permits the same illegal entry, the discrepancies can be coalesced into a single error. The example in

Figure shows the report from the cargo bag doors subsystem. In this case, six discrepancies have been coalesced into one illegal entry error and one restricted entry error (which is a by-product of the illegal entry).

Wizard-style help in resolving problems

A wizard-style interface is being developed to guide the engineer through the discrepancies between the two state charts. For each of these, AutoSteve can simulate the circuit and display the current flow using the engineer's ECAD package. Alongside this, it can show the specification state chart with the active states highlighted. The engineer can consider both of these, and determine which one of them is correct. In many cases, a single error can cause all of the discrepancies.

It has been noted above that if the subsystem behavior is sufficiently different to the specification, it may not be possible to map the envisionment states to the specification. Again, the wizard-style interface will show the circuit state of each envisionment state, and the specification state chart with the potential matches highlighted. This can assist the engineer in discovering the cause of the problems.

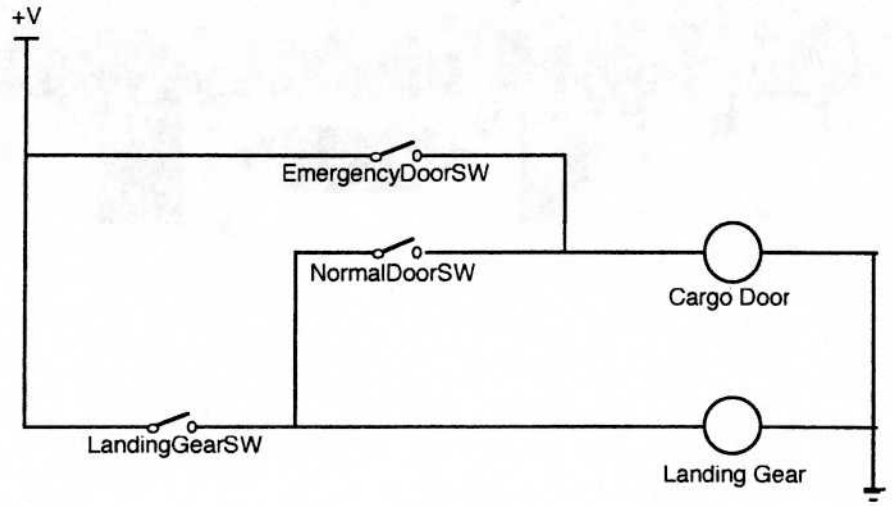

Figure 3: Cargo Bay Doors Schematic

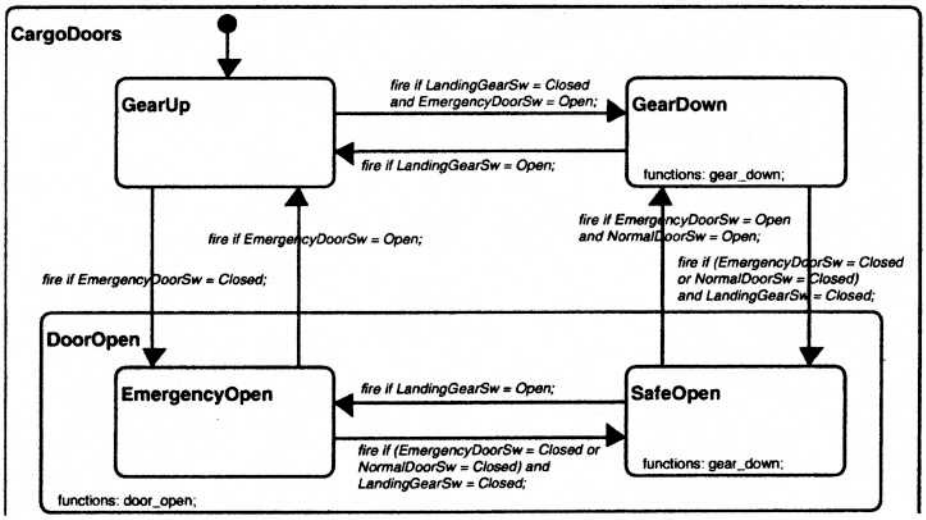

Figure 4: Cargo Bay Doors Specification

1.1. Illegal Entries to States

The following states may be entered illegally:

1. State SafeOpen, from all other states, when EmergencyDoorSw = Closed and NormalDoorSw = Closed and LandingGearSw = Open.

1.2. Restricted Entries to States[1]

Entry to the following states is restricted:

Figure 5: Extract from Cargo Bay Doors Report

Results

One application of design verification is to detect problems caused by sneak-circuits. Typically, a problem might be caused when a wire that was expected to provide current in one direction is used in the opposite direction, causing a sneak path. A good example problem is given in Savakoor et al. [9] and pictured in Figure 3, for the cargo bay doors of an aircraft. The cargo door switch should only make the cargo door open when the landing gear is down (i.e. when the aircraft is on the ground). Operating the emergency switch for the cargo doors can cause the landing gear to lower unintentionally while the aircraft is in flight if the normal door switch is also closed. Similar published examples also exist in car electrical systems.

Figure 4 shows the specification for the subsystem, with an extract from the generated report in

Figure . The report correctly identifies that the DoorOpenGearDown state (where the door is open and the landing gear is down) is illegally entered when both door switches are closed. The restricted entry problems in the other states follow from the same problem.

Limitations and Further Work

Limitations of QCAT

Ambiguities may arise in QCAT due to its qualitative representation of resistance and current, and due to its qualitative representation of time [10]. This might prevent it from being able to generate an envisionment.

State Chart Expressiveness

Local variables are not permitted in the specification state charts, so that the state charts can be statically compared. This does prevent analysis of any subsystem that cannot be expressed without local variables (for example, some circuits that use counters). In practice however, we have not found this to be a problem with those circuits used in the automotive industry.

Interface Settings

Throughout the normalization processes, the transition conditions are being compared and combined. The prototype design verification tool represents the conditions as truth tables, which means that the interface settings must be enumerations. If the interface to the subsystem could use real numbers, the manipulation of the conditions would have to be through a theorem prover, which would add another level of complexity to the analysis. However, the use of enumerated settings is consistent with the idea of qualitative circuit simulation.

Computational Complexity

Generating an attainable envisionment has an exponential complexity. Although we believe that our current system will be adequate for the majority of automotive subsystems, extending this to perform whole-car design verification will not be practicable.

One interesting line of research for whole-car design verification would be to analyze the topology of a schematic in order to generate a list of "interesting" interface settings. These could then be used generate a partial envisionment. By comparing this to the specification state chart, the system could check that the schematic does not violate the specification. Clearly, it would not be able to determine whether or not the schematic is a complete implementation of the specification, but this would have already been performed at the subsystem level. This type of analysis might provide sneak-circuit detection at a whole-car level.

Defining a state chart that specifies the behavior of a whole car would be an intimidating prospect, so we intend to consider how the individual subsystem specifications could be combined.

Conclusions

We have developed a system that can perform design verification of automotive electrical subsystems.

Features of this system include:

- It supports reusability, by allowing the engineer to re-use the qualitative models developed for FMEA, and the structural description normally entered in the ECAD tool.
- It allows engineers to specify the behavior of a subsystem using a familiar notation (state charts).
- It presents results in terms of the original specification.
- It guides engineers through the debugging process.

It successfully detects any discrepancy between the specification and the implementation, including those caused by sneak-circuits, sequence and timing errors. We intend to develop a more robust version of this prototype, and to obtain feedback from engineers using the system for real design work.

Acknowledgements

This work has been supported by the UK Engineering and Physical Sciences Research Council (grant number GR/L20542), the Ford Motor Company Ltd., Jaguar Cars, Transcendent Design Technologies and Integral Solutions Ltd.

References

1. C. J. Price, D. R. Pugh, M. S. Wilson, N. Snooke, "The Flame System: Automating Electrical Failure Modes & Effects Analysis (FMEA)", *Proc. Ann. Reliability and Maintainability Symp.*, 1995 Jan pp 90-95.
2. C. J. Price, "Effortless incremental design FMEA", *Proc. Ann. Reliability and Maintainability Symp.*, 1996 Jan pp 43-47.
3. C. J. Price, N. Snooke, D. R. Pugh, J. E. Hunt, M. S. Wilson, "Combining Functional and Structural Reasoning for Safety Analysis of Electrical Designs", *Knowledge Engineering Review* vol 12(3), 1997 pp 271-287.
4. C. J. Price, N. Snooke, J. Landry, "Automated Sneak Identification", *Engineering Applications of Artificial Intelligence*, vol 9(4), 1996 pp 423-427.
5. C. J. Price, "Function-Directed Electrical Design Analysis", *Artificial Intelligence in Engineering*, vol 12, 1998 pp 445-456.
6. Y. Iwasaki, B. Chandresekaran, "Design Verification through Function- and Behavior-Oriented Representations", *Proceedings Artificial Intelligence in Design Conference*, eds. J. Gero & F. Sudweeks, 1992, pp 597-616.
7. V. Sembugamoorthy, B. Chandresekaran, "Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems", *Experience, Memory and Reasoning*, eds. Kolodner & Riesbeck, Lawrence Erlbaum, Hillsdale, New Jersey, 1986, pp 47-73.
8. D. Harel, A. Pnueli, J. P. Schmidt, R. Sherman, "On the Formal Semantics of Statecharts", *Proceedings 2nd IEEE Symposium on Logic in Computer Science*, 1987, pp 54-64.
9. D. S. Savakoor, J. B. Bowles, R. D. Bonnell, "Combining sneak circuit analysis and failure modes and effects analysis", *Proc. Ann. Reliability and Maintainability Symp.*, 1993 Jan pp 199-205.
10. Snooke, N.A. "Simulating Electrical Devices with Complex Behavior". *AI Communications*. 1999.