AFSPG: An Automatic Faulty SPICE Program Generation System*

Kuen-Jone Lee,Cheng-Yi Hwang
Department of Electrical Engineering
National Cheng-Kung University
Tainan, Taiwan, R.O.C.

Ying-Kun Tsao
Avionics Development Center
Institute for Information Industry
Taipei, Taiwan, R.O.C

Abstract

In this paper, we present the AFSPG: an Automatic Faulty SPICE Program Generation System, which can help the user to insert any fault into a SPICE Program such that information of faulty circuit simulation can be easily collected to help later fault analysis and diagnosis. Unlike previous work that can only model single short or break faults, we replace each faulty component by a subcircuit and hence great flexibility is achieved. The user can either uses the default fault models provided in our faulty component library or defines his own fault models. The system also provides a user-friendly, window-based interface and can be easily integrated with any other fault analysis/diagnosis system.

1 Introduction

Analog PC boards are widely used in military systems and some large scale mechanical and electrical equipment such as the rapid transportation systems. These boards are usually very expensive. If some components or ICs on a PC board fails, it is necessary to find out these faulty components, and replace them with good components in a cost-effective mothod. Traditionally, the solution to fix up a PC board is based on a brute-force method, i.e., replacing each component on the PC board with a good component one by one to find out which faulty component is causing the problem. This kind of approach is quite time consuming and usually require extensive experience.

To minimize the fixing time of an analog PC board, some previous work has tried to analyze the faulty circuits in a prior and establish a so-called *diagnosis database* in which the relations between the faulty components and the fault syndrome are recorded, usually in a tree structure[5]. With this database the fault diagnosis process can be simplified to a tree search problem.

To built the database, it is critical to implement a simulation system that can help the user to find the effects of any possible fault. Since SPICE is currently the industry standard tool for circuit simulation, it becomes clear that we need a system that can generate a faulty SPICE program based on any required fault model.

Such a concept is not new. In fact, recently a commercial product, named IsSpice which is a SPICE simulator, has implemented a fault insertion function into its system [1]. By the way of its integrated system including SPICE simulator and fault insertion function, designers can insert faults and obtain the response information of the faulty circuit easily. However in [1], only two simple fault models are defined, namely short and break. Hence for some faults that cannot be adequately described by simple short or break, the system will have some difficulty in producing an appropriate faulty SPICE program.

In this paper, we shall describe this problem in detail in Section 2 and then propose a novel method to handle this problem. Based on this method we have implemented a fault generation system called the AFSPG, an Automatic Faulty Spice Program Generation System. This window-based system allows the user to generate a faulty SPICE program with any possible fault through a user-friendly interface. It also provides a faulty component database through which the user can either uses the default fault models or define his own fault models. At the end of this paper, we list the default faulty components [3][4] in our database which are based on the failure modes defined in the Consolidated Automated Support System (CASS) proposed by the American Navy [6] .

2 Faulty Component Generation

In [1], only the faults defined on the connection of a component, i.e. **open** or **short**, are taken into consideration. It uses the following general method to generate the faulty components. **If a component has a short fault between its nodes A and B, a small resistor will be cascoded with this component between nodes A and B, and if a component has an open fault at its node A, a huge resistor between node A and this component will be inserted.** Figure 1 shows these two fault models.

For most cases, this method will work well. However, if one wants to model the faults inside the components or define his own fault model, this method become

*This work was supported by the Institute for Information Industry under contract number 87R51.

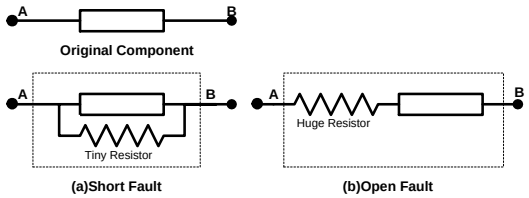

Figure 1: Faulty Components defined in [1].

inadequate. Next we use a BJT to illustrate this probelm.

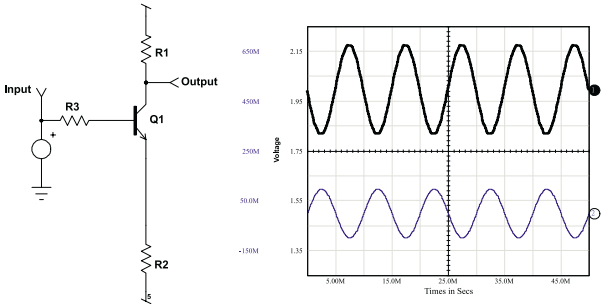

Figure 2: Sample circuit and its simulation result. The upper wave is output and the lower wave is input.

When a BJT has an BE **open fault**, there will be at least three possible cases: 1) The connection of the base node in the BJT is broken. 2) The connection of the emitter node in the BJT is broken. 3) The junction between the base and the emitter nodes in the BJT is broken. Clearly the mothod in [1] cannot handle all these three cases because a single break fault model is used. Though one may argue that Case 1 and Case 2 can be handled by defining the breaks on the base and emitter nodes, respectively, similar method, however cannot correctly handle the third case, which is an fault defined inside the component. In Figure 3, we show the circuit generated by [1] after inserting a BE Open fault. It actually inserts a huge resistor to the base node and another to the emitter node of a BJT, hence causing both the BE Open fault and BC Open fault. The simulation result shows that the output is fixed at the power supply voltage. However, since a BJT is composed of two opposite diodes[2]: BE junction and BC junction, if only the BE junction is broken, the BJT should act like a diode (BC junction). Consequently, in the third case, BC Open fault should not appear, and the faulty circuit in Figure 3 is not appropriate.

To model this fault correctly, we can use a new faulty component as shown in Figure 4. We can cascode a small resistor with BE nodes to make BE short, which eliminates the effect of BE junction, and insert a huge resistor between the node E of Q1 and R2 to make BE open. The simulation results of our faulty component is given at the right side of Figure 4, which shows that this faulty component actually acts like a diode.

From the above discussion, it is obvious that using only a simple short or break model to generate faulty components for all kinds of fault free components

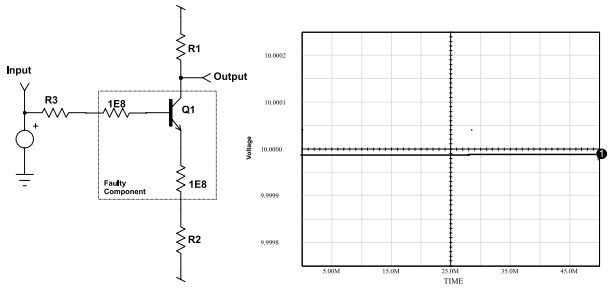

Figure 3: Fault model for BE open fault defined in [1] and its simulation result.

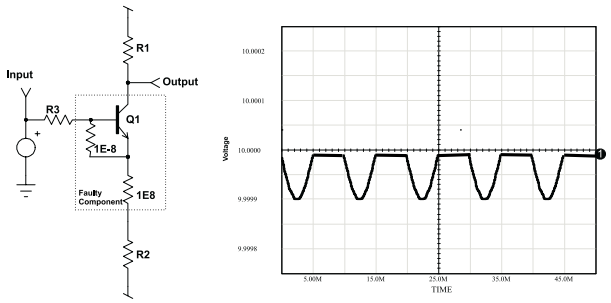

Figure 4: Our faulty component for BE open fault and its simulation result.

will have some difficulty in producing appropriate faulty SPICE programs. In this work, we propose a novel method to overcome this problem. Our basic idea is very simple: using a sub-circuit, i.e., the .SUBCKT construct defined in SPICE, to define a faulty component and replace a fault free component with this .SUBCKT. More formally, we use a sub-circuit to encapsulate a faulty component into a formal format. We define a **faulty component template format** as shown in Figure 5 to encapsulate any faulty component.

```
.SUBCKT FaultyComponentName [node list]
FaultFreeComponentName [node list] <org>
###################
# Editable Region #
###################
.ENDS
```

Figure 5: Faulty component template format.

In this format, `[node list]` contains the nodes in the original fault free component. `<org>` is a field to keep the parameters of the original component so that the faulty component can inherit its original properties defined by these parameters. The important of this arrangement will become clear through the following example. The editable region can be edited by the user so that he can create his own faulty component templates according to his own fault model.

For example, assume a user wants to insert an open fault into the capacitor C1 in the simple RLC circuit shown in Figure 6. He can define the COpen faulty component template as shown in Figure 7. According to this template, our generation insertion function can generate a faulty component by replacing the `<org>` tag

with the parameters of the original capacitor. After the faulty component is generated, the insertion function will replace "C1 3 0 1.0824NF" in the original circuit with "X_AFSPG1 3 0 COpen1", and append the generated faulty component at the end of circuit, as shown in Figure 8, where the parameter 1.0824NF appears in the first component of the .SUBCKT structure. Clearly with our method, any fault model can be defined in the sub-circuit structure, no matter how complex it is. For example, one may replace a transistor with its small-signal analysis model with some components of this model modified or some new components added to model any fault. Since all the modifications are made in the sub-circuit, one need not worry about whether any new component will cause any conflict in naming or numbering a new node or new component.

```
Simple RLC Circuit
.AC DEC 20 100 100meg
.TRAN .5u 50u
.PZ 2 0 4 0 VOL PZ
C1 3 0 1.0824NF
C2 40 1.5307NF
R1 1 2 1OHM L1 2 3 .38268UH
L2 3 4 1.5772UH
V1 10 AC=1 PWL 0S 0V 1US 0V 2US 1V 20US 1V 20.1US 0V
.END
```

Figure 6: Sample faulty-free SPICE program

```
.SUBCKT COpen 1 2
C1 1 3 <org>
R1 2 3 1E8
.ENDS
```

Figure 7: COpen faulty component template

```
Simple RLC Circuit
.AC DEC 20 100 100meg
.TRAN .5u 50u
.PZ 2 0 4 0 VOL PZ
X_AFSPG1 3 0 COpen1
C2 40 1.5307NF
R1 1 2 1OHM L1 2 3 .38268UH
L2 3 4 1.5772UH
V1 10 AC=1 PWL 0S 0V 1US 0V 2US 1V 20US 1V 20.1US 0V
#####Fault Module Added By Afspg System######
.SUBCKT COpen1 1 2
C1 1 3 1.0824NF
R1 3 2 1E6
.ENDS
.END
```

Figure 8: Sample faulty SPICE Program

3 Implementation

3.1 AFSPG System Architecture

We have implemented a window-based integrated fault insertion system, called the Automatic Faulty SPICE Program Generation system (AFSPG) . Its functional blocks are shown in Figure 9 and the function of each block is described below.

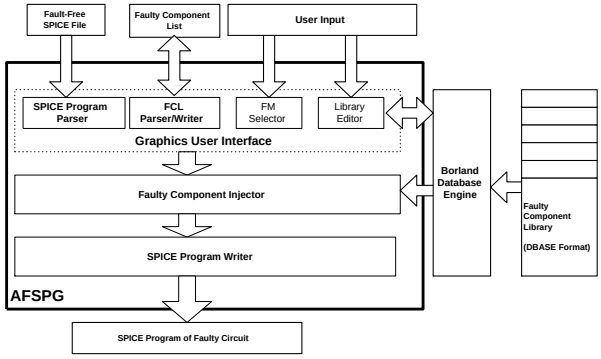

Figure 9: Functional Block of AFSPG System

- SPICE program parser:
  First the SPICE Program Parser will read a fault free SPICE program and parse each component of it. Then it will maintain a list of the components it parsed and display the list in the graphic user interface, as shown at the lowest-left region of Figure 10. In this region, we use a tree view to display the hierarchical relation of the SPICE program and its components. The top node, named "Projects", is the root node. The child node of the top node, named "test.CIR", is the SPICE program currently being opened in the AFSPG system. The child nodes, "V1", "D1" etc., of the node "test.CIR" are the components contained in this circuit file. Each of these components, may have some child nodes which indicate the available failure modes corresponding to the component. Assume we have defined two failure modes corresponding to a capacitor, then every capacitor will has two child nodes, e.g., "C1" has two child nodes "COpen" and "C-Short".

- Failure Modes Selector:
  After AFSPG had parsed the required SPICE program, the user can select the faults that he wants to insert, and if necessary, modify the parameters of the faulty components, as shown in Figure 10, where the fault selected is "COpen" fault of C1. The exclamation mark in front of the fault indicates that the fault has been selected. The lowest right region displays the faulty component template corresponding to the COpen fault, and its contents is editable. The designer can modify the default parameters or complete replace the faulty component template with his own model through this Faulty Module window.

- Faulty Component Injector:
  After a designer has selected the faults that he wants to insert into the fault free circuit, the Faulty Component Injector will generate the faulty component according to the faulty component template

corresponding to the original fault free component. Then it will replace the original components with the faulty components it has generated. Note that this modification will only apply to the current component, i.e., C1. All other capacitors will no be affected.

- Library Editor:
  Our system also provides a faulty component library editor, as shown in Figure 11. This DBASE based library maintains some essential fields which a faulty component template needs. Through this library, our system can figure out the component each fault corresponding to and display it in the tree view as mentioned. In addition, a designer can maintain his own faulty component templates according to his own fault model. Note that if the user edits his fault model by the Library Editor, the modified template will apply to all components that refers to this fault model.

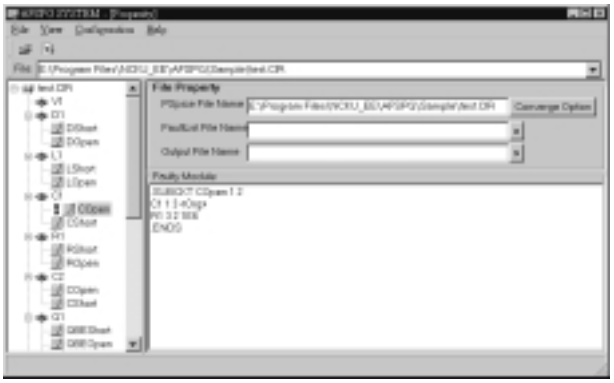

Figure 10: Snapshot of AFSPG

Figure 11: Snapshot of Library Editor

4 Conclusion

In this paper, we present the AFSPG System: a window-based system that allows a user to insert any faults into SPICE programs. Based on this system, one can define his own fault model or use the provided default fault model to create appropriate faulty SPICE program. Our system has now been incorporated into the Intelligent Diagnosis and Test System (IDTS) developed by the Institute for Information Industry and is a key component in establishing the Fault Diagnosis Database for IDTS. To complete this paper, a list of default fault models for simple analog components based on the CASS standard is given at the end of this paper.

References

[1] "IsSpice4 User's Guide," *Intusoft Corporation* 1996.

[2] Sedra Smith, "Microelectronic Circuits, Third Edition," 1992.

[3] Paolo Antognetti, Giuseppe Massobrio, "Semiconductor Device Modeling with SPICE," 1988.

[4] E.A. Amerasekera, D.S. Campbell, "Failure Mechanisms in Semicondctor Device," 1987.

[5] Ruey-Wen Liu, "Testing and Daigonsis of Analog Circuits and Systems," 1991.

[6] "CASS Red Team Package data item DI-ATTS-80285, Fig. 1 SRA/SRU Fault Accountability Matrix Table, pp. 11."

Table 1: Our Faulty Component List

Component Type	Faulty Component Name	Implementation
R [Resistor]	RShort	Change R Value to 1E-6
R [Resistor]	ROpen	Change R Value to 1E6
L [Inductor]	LShort	Cascode L With R(1E-6)
L [Inductor]	LOpen	Cascade L With R(1E6)
C [Capacitor]	CShort	Cascode C With R(1E-6)
C [Capacitor]	COpen	Cascade C With R(1E6)
D [Diode]	DShort	Cascode D With R(1E-6)
D [Diode]	DOpen	Cascade D With R(1E6)
Q [BJT]	QBEShort	Cascade BE With R(1E-6)
Q [BJT]	QBEOpen1	Cascade BE With R1(1E-6) Cascode E With R2(1E6) [Junction BE break]
Q [BJT]	QBEOpen2	Cascade E With R(1E6) [Node E Break]
Q [BJT]	QBEOpen3	Cascade B With R(1E6) [Node B Break]
Q [BJT]	QCEShort	Cascode CE With R(1E-6)
Q [BJT]	QCOpen	Cascade C With R(1E6)
J [JFET]	JSDOpen	Cascade S With R1(1E6) Cascade D With R2(1E6)
J [JFET]	JSDShort	Cascode SD With R(1E-6)
J [JFET]	JGOpen	Cascade G With R(1E6)
J [JFET]	JGSShort	Cascode GS With R(1E-6)
J [JFET]	JGDShort	Cascode GD With R(1E-6)
M [MOSFET]	MSDOpen	Cascade S With R1(1E6) Cascade D With R2(1E6)
M [MOSFET]	MSDShort	Cascode SD With R(1E-6)
M [MOSFET]	MGOpen	Cascade G With R(1E6)
M [MOSFET]	MGSShort	Cascode GS With R(1E-6)
M [MOSFET]	MGDShort	Cascode GD With R(1E-6)