THE IMPACT OF WORKLOAD ON THE DEPENDABILITY OF MICROPROCESSORS USED IN CONTROL APPLICATIONS

BY

RAMAN V. KRISHNAN

B.Tech., Indian Institute of Technology, Madras, 1990

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1996

Urbana, Illinois

ABSTRACT

It is well-known that the workload applied to a computer system has an impact on its dependability. This dependence can be due to many factors. It is also believed that workload has an effect on the failure rate of integrated circuits. This thesis investigates this claim, considering the effect the execution of a particular program has on the permanent failure rate of a microprocessor. In particular, we develop a methodology for evaluating the failure rate due to electromigration of non-pipelined microprocessors used in control applications, taking into account the program that executes on the chip. We do this by first performing a detailed logic-level simulation of instructions on the chip, building a workload *signature* for each instruction that gives the mean interswitching time for all nodes on the chip. We then use these signatures to determine the mean interswitching times at each node for particular programs and use the interswitching times to predict the mean time to failure for the chip when executing the program. We illustrate the methodology via a study of programs executing on a HS1602 microprocessor that has been used in avionic flight control applications. The results show that our methodology can be used to predict the failure rate – considering the program executed

– of non-pipelined microprocessors, and that the overall failure rate of the chip, due to electromigration, does indeed depend on the workload applied to the chip.

ACKNOWLEDGEMENTS

I express gratitude to my advisor, Professor William Sanders, for his invaluable insight, and technical advice and support on this project. I wish to thank Professor Ravishankar Iyer, who with Professor Sanders, created and molded this interesting research problem.

I also would like to thank Professor Gwan Choi of Texas A&M University for providing useful information about the HS1602 processor and the SPLICE simulator. Thanks are due to my colleagues Haiming Jin and Aad Van Moorsel for many interesting discussions.

Finally, I thank my wonderful parents, brother and sister for their encouragement and support. This thesis is dedicated to them.

TABLE OF CONTENTS

Page

1. INTRODUCTION 1
   1.1 Related Research 3
2. ENVIRONMENT/METHODOLOGY OVERVIEW 6
3. ENVIRONMENT COMPONENTS 10
   3.1 Logic-Level Simulator 10
   3.2 Trace Processor 13
   3.3 Instruction/Signature Analyzer 13
   3.4 Analytical Electromigration Model 14
4. EXAMPLE STUDY 19
   4.1 Target System 20
   4.2 Workload Categories 21
   4.3 Results 25
5. CONCLUSIONS AND FUTURE WORK 31
6. APPENDIX 33

REFERENCES 50

LIST OF TABLES

Table Page

4.1: Example mean interswitching times 24
4.2: Failure time (in years) of target chip for instructions 26
4.3: Failure times (in years) of the HS1602 for groups of instructions . . . 27

LIST OF FIGURES

Figure Page

2.1: Combined experimental/analytical environment. 7
3.1: Elements of the simulation and trace processing procedure. 10
3.2: Sample trace data 12
4.1: (a) Physical structure of the metal node. (b) Structure of crack. . . . 20
4.2: Block diagram of the target chip. 22
4.3: Switching *signature* for *AND* instruction. 23
4.4: Switching *signature* for *INA* instruction. 23
4.5: MTTF of chip with *COMP* mix[0.0,1.0] and *SKIP* mix [0.0,0.1]. . . 28
4.6: MTTF of chip with *COMP* mix[0.0,1.0] and *SKIP* mix [0.0,1.0]. . . 29
6.1: Program flow chart for computation of MTTF. 34

1. INTRODUCTION

As microprocessors become increasingly complex and are used in critical applications, their dependability is an increasing concern. Of particular interest is the study of the time to permanent failure of integrated circuits. Many factors can contribute to these failures, including temperature, feature size, and fabrication technology. Failure mechanisms are also varied, but according to Siewiorek and Swarz [15], 51% of failures observed in LSI standard TTL technology and 8% of failures in CMOS are due to metalization. As feature size shrinks, this effect may become even more significant. One of the key factors contributing to the permanent failures of microprocessors is the phenomenon of metalization known as electromigration, which is the migration of metal atoms as a result of very high current densities flowing through the metal.

Although it has been well-established that the type of workload applied to a microprocessor influences the manifestation of transient faults, little has been studied about the permanent faults with regard to workload issues. This thesis investigates the impact of workload on the dependability, considering the effect that the execution of a specific

software has on the permanent failure rate of the microprocessors. A methodology is developed for evaluating the lifetime due to electromigration of a non-pipelined microprocessor used in control applications taking into account the program that executes on the chip, using a combined simulation- and analysis-based approach.

In this methodology, a detailed switch-level simulation of a target chip is first performed using SPLICE [7] to acquire trace data on the switching activity for various instructions of the chip. The switching *signature* built for each instruction is then used to compute the mean interswitching times for all logical nodes of the target chip. These interswitching times are then used, together with an existing analytical electromigration model, to predict the mean time-to-failure of the microprocessor executing a program with a specific mix of instructions.

A naive way to compute the reliability of the entire IC chip as a function of workload would be to execute the actual workload using a simulator, a process that is highly expensive in time. Instead a scheme is developed wherein the *signatures* and the instruction mix of the workload are exploited to accelerate the computation of the chip reliability. The methodology is illustrated via a study of programs executing on HS1602, a microprocessor that has been used in avionic flight control applications. The results show that our methodology can be used to predict the lifetime of non-pipelined microprocessors, considering the software executed, and that the overall failure rate of the chip caused by electromigration does indeed depend, among other factors, on the workload applied to the chip.

The rest of the thesis is organized as follows. The following section discusses the related research in this area. Chapter 2 presents the overview of the experimental environment and methodology. The logic level simulation, post processing of the trace data, and instruction analyzer are described in Chapter 3. The electromigration model used in our analysis is described Section 3.4 of Chapter 3. Example results from the case study of HS1602 are presented in Chapter 4, and finally conclusions are drawn in Chapter 5.

1.1 Related Research

The phenomenon of electromigration-induced failure of integrated circuit metalization has been the subject of much past study. In particular, several electromigration models have been proposed to study the reliability of integrated circuits. An early investigation [1] of electromigration showed that the mean time to failure (MTTF) of a metal stripe under constant current density can be expressed by an empirical equation known as *Black's equation*. Based on Black's equation, a number of approaches to analyze and predict the reliability of integrated circuits due to electromigration failure mechanisms have been proposed [8, 12, 10]. For example, in [16], the lifetime of a metal stripe due to electromigration is evaluated using a simple analytical model. Several simulation-based approaches to studying the effects of electromigration have also been proposed. In particular, a method using a Monte Carlo simulation for electromigration is described in [14]. All of these works have concentrated on the effects of a steady dc current on reliability, although most VLSI designs now employ pulsed current throughout the chip.

A few research efforts have been directed toward pulsed currents. Specifically, the lifetime due to electromigration for arbitrary waveforms is projected by [10], [2], and [13]. An experiment to determine the MTTF due to electromigration with pulsed rectangular current applied to the metal line is described in [2]. These works suggest that the manner in which the current is pulsed can have a significant effect on an integrated circuit's reliability.

However, there has been no link between the models for pulsed current on the circuit-level in a microprocessor, and the workload (software) that induces current pulses at the circuit level. This is a important gap that needs to be filled if we are to understand how the workload of a microprocessor affects its reliability.

The importance of workload has been realized in the case of transient faults, however, and provides a guide for work in the area of permanent faults. Specifically, workload effects have been analyzed for manifestation of transient faults in [6, 3, 5]. In [5], for example, a fault behavior model which used workload attributes (instruction types and counts) and fault location as its primary parameters was developed.

This thesis combines work in the area of electromigration and workload-based fault analysis, considering the effect the execution of particular software has on the permanent failure rate of the microprocessors. Initial work in this regard has been done by Choi and and Iyer, who studied the permanent failure rate of of a microprocessor executing a program using importance sampling simulation [4]. They did not, however, study how differing workloads could affect a microprocessor's reliability. Thus the contribution

of this thesis is an attempt to bridge the gap between the workload issues and the phenomenon of permanent failures caused by electromigration.

2. ENVIRONMENT/METHODOLOGY OVERVIEW

This section provides a broad overview of the methodology used in our evaluation, while the subsequent sections detail the procedure followed. The graphical representation of the methodology is presented in Figure 2.1. Boxes in the figure represent components that act on data, and ovals represent data that are acted on. The main components of the environment are 1) a logic-level simulator, 2) a trace processor, 3) an instruction mix analyzer, and 4) an analytical electromigraton model. Together, they can predict the impact a particular program will have on the MTTF of a microprocessor.

A key component of the environment is a circuit simulator, which allows the logical simulation of an entire microprocessor, support chips, and program (stored in ROM or RAM) executing on the microprocessor. The simulator provides the mechanism to translate a program, written in the machine language of the target microprocessor and stored in the ROM or RAM support chips, into a sequence of current pulses on the microprocessor. A naive way to obtain the frequency of the current pulses at each node would be to execute the program directly using the simulator, recording the switching

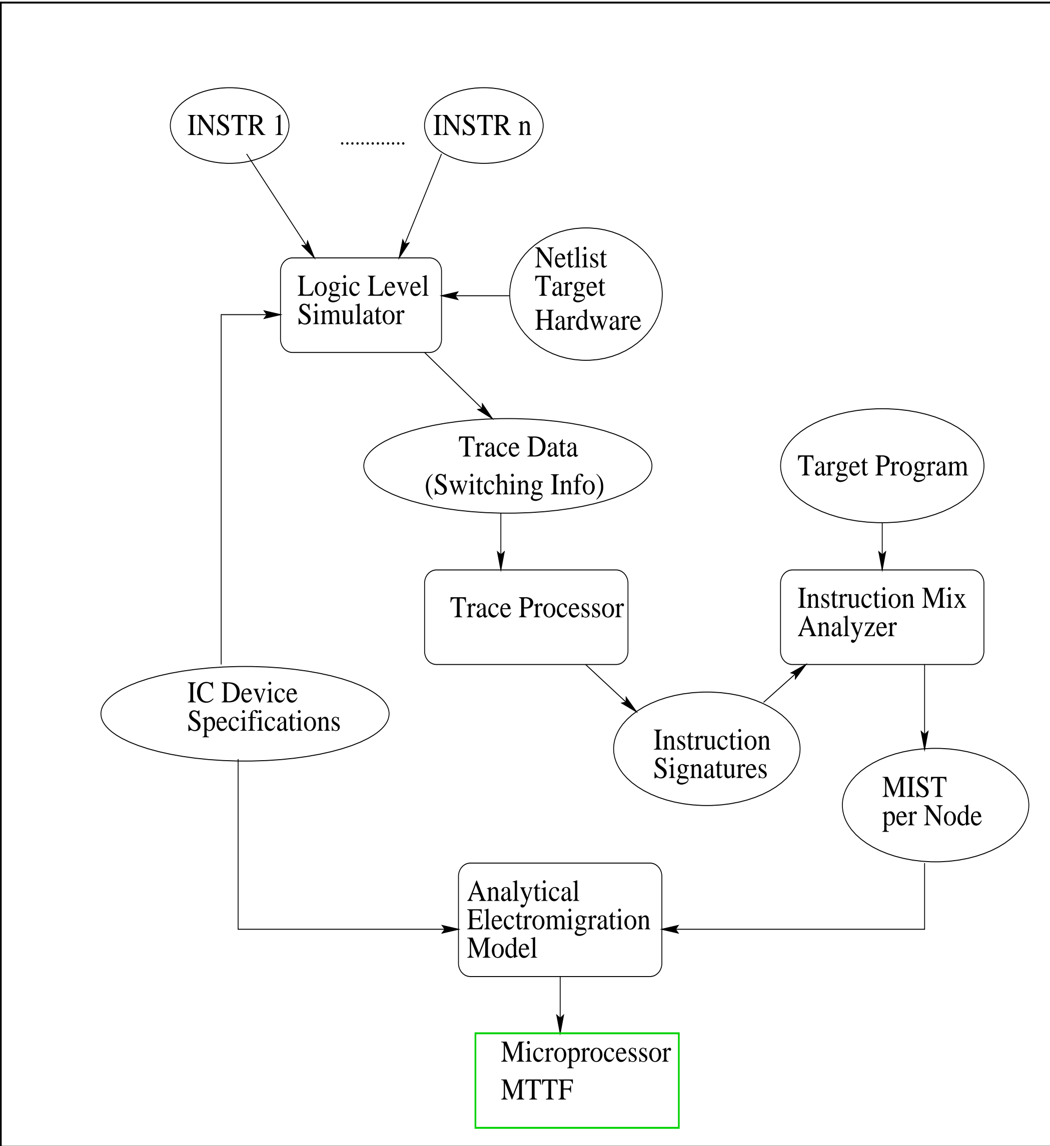

Figure 2.1: Combined experimental/analytical environment.

activity at each node. This would, however, be extremely slow since the simulation must be done at the logic level. Instead, we obtain the frequency of current pulses for particular instructions individually by executing them repeatedly on the chip. While the state of the microprocessor (dictated by the instructions surrounding the instruction in question) could potential affect the switching frequency of nodes during the program's execution execution, the effect should be minimal for simple non-pipelined microprocessors such as the HS1602 used in our example study.

A tracing facility in the simulator is used to monitor the switching activities of all internal nodes defined in the netlist description of the target chip. The netlist, in addition to the microprocessor description, contains information about the nodes that are being probed for trace. The trace data obtained are voluminous, consisting of the time of each logic-level state change, the hierarchical node name that experiences the change, and the new logic level. They are then processed by the trace analyzer to determine the "signature" for each instruction, giving the mean interswitching time at each node in the microprocessor for the instruction.

These mean interswitching times, together with the target program, serve as input to the instruction mix analyzer. The instruction mix analyzer first calculates the frequency of each instruction in the program. It then calculates the mean interswitching times at each node from the instruction mix of the program and the instruction signatures generated by the trace processor. These interswitching times form a "signature" for the

program, which defines the mean interswitching time for each node on the chip, when executing the program.

The program signature, together with the physical specification of the chip are fed to an analytical electromigration model [16]. This model estimates the electromigratory effects in various metallic nodes of the target chip assuming specific operating conditions, device parameters, and fabrication technology. The electromigration model first computes the time to failure for a constant dc current flowing through the metal lines. It then uses the interswitching times, computed by the instruction mix analyzer, and the switching times of the chip, defined by its physical parameters, to compute a "duty factor" for each node. This duty factor is then used to compute a MTTF for the node, taking into account the workload. Finally, by assuming independent and exponential failure rates for all nodes, we compute the MTTF of the entire chip as a function of the instructions it is executing.

The sections in the following chapter describe the evaluation environment in more detail, making clear assumptions used in the environment and illustrating its use on an example microprocessor used in flight control applications.

3. ENVIRONMENT COMPONENTS

The simulation and analytical environment used in our study consists of the following components.

3.1 Logic-Level Simulator

Translation of instructions in the instruction set of the target microprocessor to switching frequency at nodes in the chip is done using SPLICE, as shown in Figure 3.1. As shown in this figure, the netlist of the microprocessor, support chips, and RAM

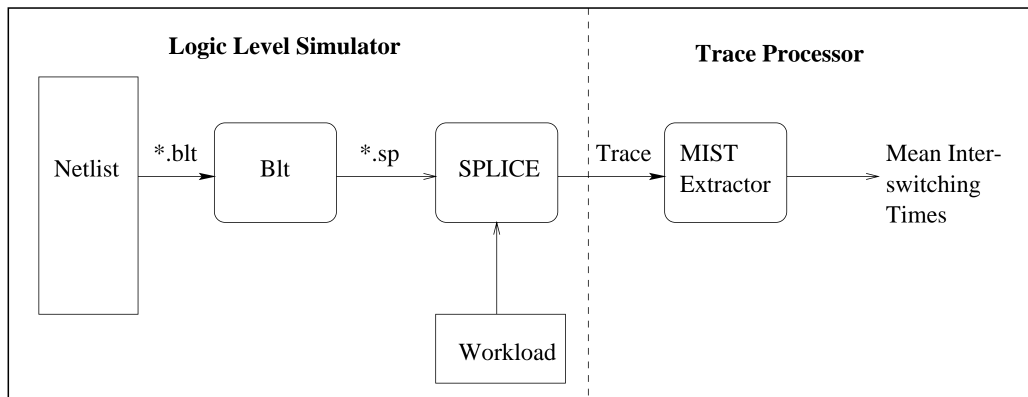

Figure 3.1: Elements of the simulation and trace processing procedure.

and/or ROM containing the target program at the functional level, in a hierarchical form, is input to a translation program, called Blt, which translates the description from BLT format to sp format. The user of the environment defines the workload as instructions in RAM or ROM that will be executed on the target microprocessor. The sp format contains timing information for each internal node. The description of the system in sp format is then fed to the simulator.

An evaluation of the switching behavior of each instruction is then carried out by filling the RAM with multiple instances of the instruction, which is implemented as an input file to the simulator. The instruction is then executed multiple times, under the command of a set of monitor programs in ROM that initiates the execution of the sequence of instructions. By repeated execution of an instruction placed in RAM, the simulator creates a trace, using the monitoring facility in SPLICE, that gives the time each node in the processor changes state ($0 \rightarrow 1$, or $1 \rightarrow 0$). Sample trace data, obtained with the simulator used in our example study, are shown in Figure 3.2. In this figure, the first column represents the time of the event in nanoseconds, the second column is the node number that experiences the event, the third column is the new logic level, the fourth column is the logic strength (not used), and the last column is the hierarchical node name, corresponding to the node number. These raw data are written to a disk file to be processed by the trace processor.

```
1      3459   0      9      hs1602_p64_d1_i3
1      3452   0      9      hs1602_p63_d1_i3
1      3445   0      9      hs1602_p61_d1_i3
1      3438   0      9      hs1602_p60_d1_i3
1      3431   0      9      hs1602_p59_d1_i3
...
2      3682   1      9      hs1602_p51_cp
2      3683   0      9      hs1602_p51_cn
2      3706   1      9      hs1602_pa3_cp
2      3707   0      9      hs1602_pa3_cn
2      3720   1      9      hs1602_pa6_cp
2      3721   0      9      hs1602_pa6_cn
...
641    782    1      7      hs1602_m1_ca_al_m2_m2_u12_i1
641    857    1      7      hs1602_m1_ca_al_m3_m2_u12_i1
641    932    1      7      hs1602_m1_ca_al_m4_m2_u12_i1
641    1008   1      7      hs1602_m1_ca_al_m5_m2_u12_i1
641    1089   1      7      hs1602_m1_ca_al_m6_m2_u12_i1
641    1164   1      7      hs1602_m1_ca_al_m7_m2_u12_i1
...
```

Figure 3.2: Sample trace data

3.2 Trace Processor

The trace processor is executed on the trace data to collect the important information from the trace. In particular, it collects the number of times each node switched during the execution of the target program and the mean interswitching time for each node. It is implemented using an *awk* script and functions by detecting each change of logic level and the time of the change for each node in the microprocessor being monitored. In doing so, it constructs a "signature" for each instruction, giving a graphical representation of the mean interswitching time vs. node number being monitored. For the example microprocessor studied (see the next section), the signatures of different instructions varied significantly. As will be seen, differences in signatures will lead to differences in the MTTF of the microprocessor when executing different mixes of instructions.

3.3 Instruction/Signature Analyzer

After obtaining the signatures for each instruction, the mean interswitching time for each node in the microprocessor given a particular target program is determined using the instruction/signature analyzer. In particular, the instruction/signature analyzer first computes the relative frequency of each instruction in the target program. Since we consider microprocessors used in control applications, we envision that the processor runs a single program in an infinite control loop, which never terminates. Determining an appropriate frequency would be much more difficult for a microprocessor intended for general purpose usage and is beyond the scope of this thesis.

The analyzer then computes the mean interswitching time at each node by weighting the interswitching time calculated on a per-instruction basis by the relative frequency of the instruction. This is valid since we are considering non-pipelined microprocessors, where the state of the processor (and hence its switching behavior) is substantially determined by individual instructions, rather than by a sequence of instructions or the internal state of the processor. The output of this stage of the processing is the mean interswitching time, for each node, for the target program. The following section details the electromigration model used in our analysis.

3.4 Analytical Electromigration Model

The mean interswitching time, for each node, together with the integrated circuit device parameters (e.g., operating voltage, line resistance and capacitance, feature size) serve as input to the module that determines the MTTF for the microprocessor, given a particular workload. This is determined in two steps, for each node in the microprocessor. First, the MTTF is computed using an existing analytical electromigration model which assumes a dc current. Second, the result is corrected to account for the mean interswitching time at each node using results in [17] that relate the MTTF in a node subjected to dc current to one driven by an arbitrary wave form. Each of these two steps is described in more detail in the following.

In particular, the analytical model used to compute the MTTF of the internal nodes treats the growth of a discontinuity quantitatively, considering grain-boundary electromigration and Joule heating around a crack-shaped void that grows until a failure occurs. In doing so, it assumes that:

1. Metal lines are of a uniform length and width overlaying an insulator slab, carrying current densities of the same order.

2. The bottom interface is maintained at a constant temperature T. This configuration of the metal stripe closely mimics the metalization stripe on an integrated circuit chip that is mounted on an effective heat sink. The temperature rise ΔT_0, due to Joule heat internally generated in the metalization, causes heat flow through the thermal impedance of the substrate to the heat sink.

3. Electromigration occurs only on grain boundaries and creates vacancies forming a crack which grows perpendicular to the current flow.

4. The initial current density is a constant, given by j_i.

5. The effects due to variation of thermal coefficient of the metal α and the possible melting of the stripe at the later stages of crack growth are negligible. This assumption is shown to have no major effect on the lifetime of the metal stripe [16].

6. The temperature dependence of q in (3.1) has no significant effect on the lifetime of the metal stripe, and its contribution is accounted for by using the value of q that is pertinent to the initial stripe temperature.

7. The growing void is immobile.

Given these assumptions, the MTTF, assuming dc current, is given in [16] as

$$MTTF_{dc} = \frac{cWKT}{\delta N_a q \rho j_i D_0 exp(-\frac{\Delta H}{KT}) 2[1 + (\frac{0.2\Delta H}{KT^2})\Delta T_0]} \tag{3.1}$$

where:

K = Boltzmann's constant

N_a = Number of atoms per cubic centimeter

δ = Effective grain width

ΔH = Activation energy of the metal

c = Effective crack width

W = Metal width

q = Positive electron charge

T = Steady-state ambient temperature

ΔT_0 = Initial self-heating

D_0 = Diffusion coefficient

ρ = Ohmic resistivity of the metal

Equation (3.1) is applied to each node in this microprocessor to compute the MTTF given a dc current. These MTTF's, assuming the flow of a direct current, then are used to compute the MTTF, given the mean interswitching time per node calculated by the instruction/signature analyzer, for each node. The relationship given between these times is given by the duty factor of the node, which is the fraction of the time the node is experiencing current as determined by the mean interswitching time for the node. More precisely, the duty factor r of a node is given by

$$r = \frac{switching_time}{switching_time + mean_interswitching_time} \tag{3.2}$$

where the switching time is determined by device parameters for each node. In doing so, we assume that the switching time of the node during charging and discharging of the

node capacitor is identical and is computed approximately as $5RC$, where R is the node resistance and C is the node capacitance.

Given the duty factor, the MTTF for the node, considering its mean interswitching time, is determined using the following equation, which was determined experimentally by [13]

$$MTTF_{Pulsed_dc} = \frac{MTTF_{dc}}{(r)^2}. \tag{3.3}$$

Implicit in the above equation is the assumption that the vacancy generation rate is proportional to a positive power of the current density [13].

Given the MTTF for each node, considering its mean interswitching time, the MTTF for the microprocessor is obtained given some assumptions about the distribution of the time to failure for a node and the dependence between nodes. In particular, if we assume (as is commonly done) that node failure times are independent and exponentially distributed, we can easily compute the minimum time to failure of a node in the microprocessor. In this case, the failure rate (1/MTTF) for the microprocessor will be the sum of the failure rates for the individual nodes, and itself be exponentially distributed.

As with all models, the electromigration model used in the environment has limitations, but we believe that these are reasonable given its intended application. In particular, the model used in our analysis applies to long stripes and neglects the temperature gradients at stripe termination points. However, it is observed that many long stripes commonly fail in regions away from the terminal points where the initial temperature gradients exist [11]. The model also disregards the positive feedback that would exist in

the crack-affected area and which may accelerate the rate of crack growth and shorten the metal lifetime.

4. EXAMPLE STUDY

In this chapter we present and discuss the example results obtained. To illustrate the use of our methodology, we consider the evaluation of a microprocessor used in flight control applications under a wide variety of application programs. The intent is not to determine an absolute measure of the mean time to failure of the chip executing a particular workload, but instead to investigate the sensitivity of the MTTF to differing workloads.

To do this, we consider a custom-designed CMOS microprocessor, the HS1602 [9], used in the avionic industry for control applications. A netlist consisting of 3927 internal nodes of the target chip at the logical level was used to simulate the activity of the nodes inside the chip as it executes various instructions. The SPLICE simulation is performed at a resolution of 1 ns under operating conditions of 5V and a constant heat sink temperature of 500° K. Most instructions of the target chip are executed for 5000 ns of real-time, ensuring multiple execution cycles of the same instruction. A few branch

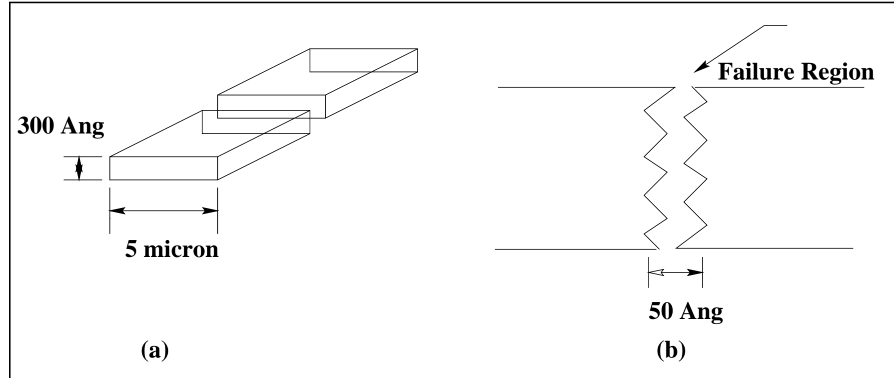

Figure 4.1: (a) Physical structure of the metal node. (b) Structure of crack.

instructions (e.g., BSA) could not be made to execute multiple cycles and hence are not considered in the experimental analysis.

The electromigration model (3.1) used in this analysis supposes uniform dimensions for all internal nodes. An example of the metal line shown in Figure 4.1 measures 5μ in width and approximately 300 A° in thickness. The metal stripe inside the target chip is assumed to be aluminum, with an intrinsic diffusivity coefficient of 1e-4 cm^2/s and a specific resistance ρ of 2.83e-6 Ωcm. The activation energy of aluminum in (3.1) ΔH is assumed to be 0.7 eV.

The following section describes the target system in detail.

4.1 Target System

The target system used in this study is a custom-designed microprocessor that forms a part of the control circuitry in jet engines. The HS1602 microprocessor, manufactured by Hamilton Standard, is currently used in commercial aircrafts, including the Boeing-747. The system is provided with two channels of control consisting of identical processing

elements. The microprocessor we study is used to control engine functions such as fuel flow, the temperature, the engine speed and external inputs such as speed and navigational parameters. These inputs are sampled in real time by the microprocessor and are updated into the RAM roughly every millisecond for further processing. The chip activity is controlled by the code resident on the ROMs that form a part of the entire control system. The various components of the overall system include microprocessors, memory units, UART transmitter and receiver units, a multiplexer, I/O gate array chips, A/D and D/A converters, and a Watchdog unit [9]. Only the microprocessor is the focus of our simulation and study. Figure 4.2 shows the microprocessor in greater detail.

As is seen from the block diagram, six functional units form the core of the microprocessor. While the Arithmetic Logic Unit (ALU) is itself capable of performing double precision arithmetic, its operations are controlled by a control unit consisting of several registers and combinational logic. The decoder unit decodes the I/O signals while a countdown unit drives the chip-wide clock signals. A significant feature of the target chip is the provision for detecting faults. The Watchdog unit, by resetting the processor in the event of parity errors or a timeout from a software application, provides the necessary fault detection mechanism.

4.2 Workload Categories

As per our aim of studying the sensitivity of the MTTF on the workload, we study a range of programs with differing “mixes” of instructions, rather than a single program

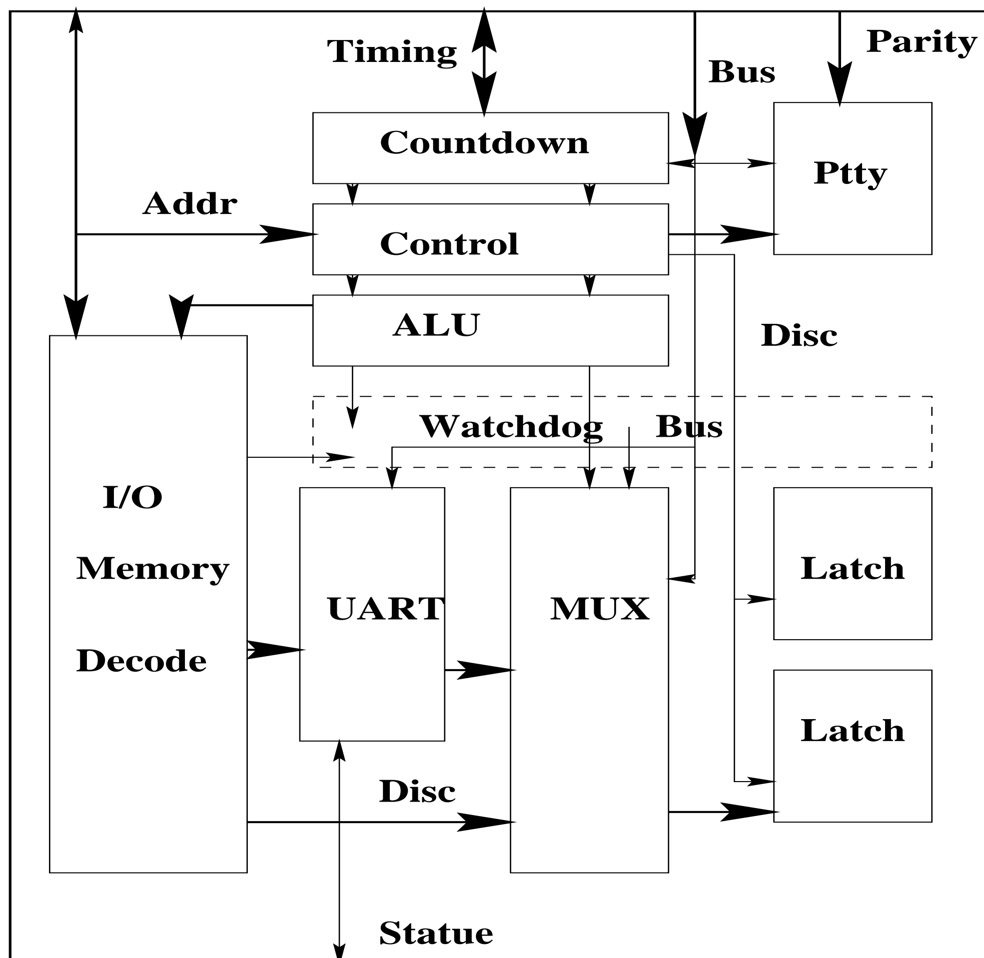

Figure 4.2: Block diagram of the target chip.

as prescribed by the methodology described in the previous section. In particular, we categorized HS1602 instructions into groups that have similar signatures and hence would have similar effect on microprocessor reliability. For instance, all skip instructions have similar signatures and hence contribute to the lifetime of the chip in a similar way. To illustrate the differences in the signatures among different instructions, consider Figures 4.3 and 4.4 which show the switching signatures for a logic instruction (AND) and an IO instruction (INA), respectively. The x-axis in the plots represents the node number,

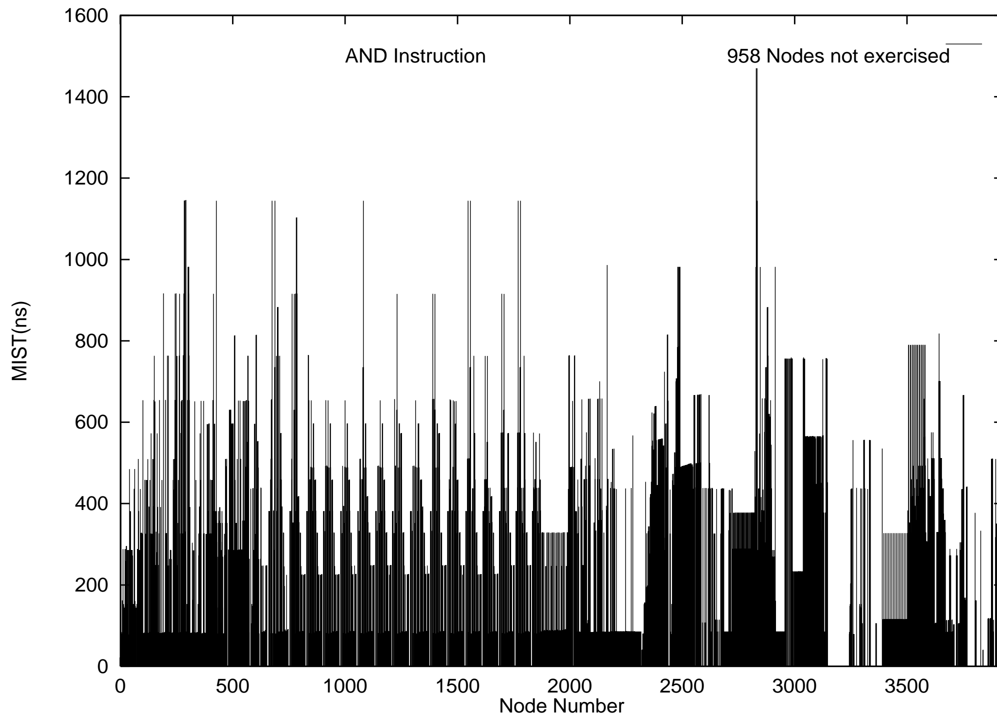

Figure 4.3: Switching *signature* for *AND* instruction.

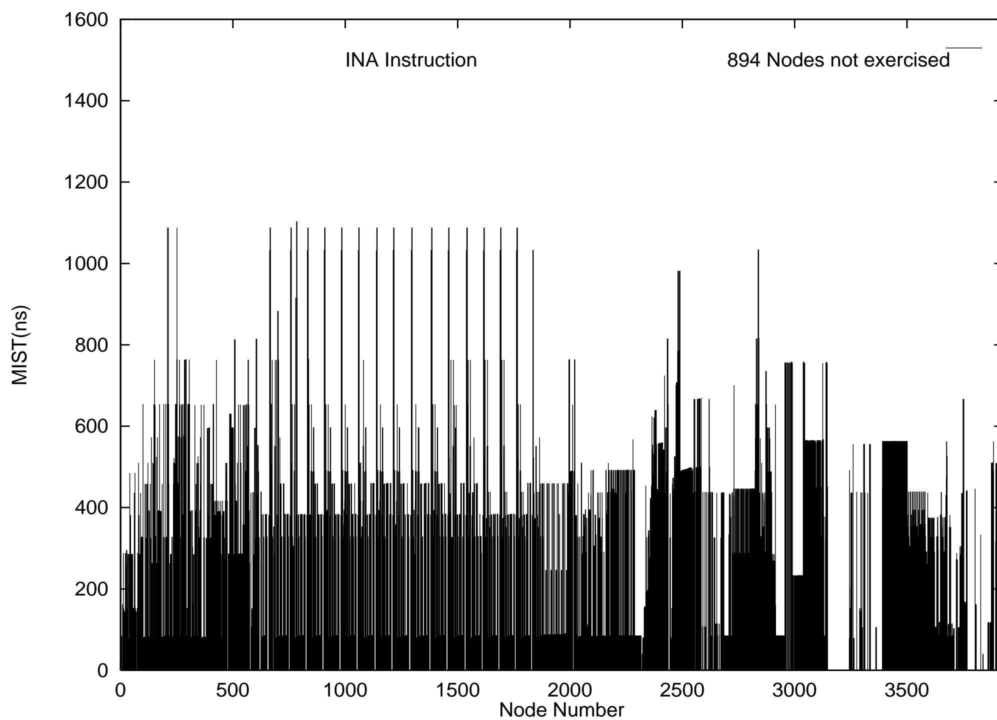

Figure 4.4: Switching *signature* for *INA* instruction.

Table 4.1: Example mean interswitching times

Instruction Name	Internal Node Number	Mean Interswitching Times (in ns)
MUL	38	319.0
SDA.B	258	650.7
CLAB	55	79.7
SOA	823	653.3
NOP	298	153.2
CLA	305	785.4
LLR	24	152.9
IAB	2799	306.8
INA	3839	40.6
AND	515	272.7

while the y-coordinates are the mean interswitching time obtained for the corresponding node numbers. Due to their inactivity, 958 internal nodes for *AND* instruction and 894 nodes for *INA* instruction were not exercised. Note that these two instructions stress the internal nodes quite differently. To illustrate the diversity of interswitching times, consider Table 4.1, which gives the mean interswitching times for various nodes executing particular instructions. Note that the interswitching time can vary greatly, depending on the instruction and node.

To build the workload for our example, we grouped the instructions into three categories, namely *Skip*, *Computation*, and *IO*. The grouping was picked because of the similarity of signatures within each group and because instructions within the group perform similar functions. Of course, the actual number of categories of instructions for a generic microprocessor would vary depending on the range of instructions supported by

the microprocessor. The MTTF values for instructions supported by HS1602 microprocessor are tabulated in Table 4.2. Table 4.3 shows the assignment of each instruction to a group. While the particular function of each instruction is not important, it is important to note that the skip instructions implement conditional and unconditional branches, the computation instructions implement logical and arithmetic computation, and the IO instructions provide the method to move data to/from input/output ports. For the example study, we consider each group of instructions as a single instruction, with a signature that is the average of the individual signatures of each of the instructions in the group.

Although not needed to determine the microprocessor MTTF for a given workload, it is interesting to observe the MTTF of the microprocessor if it were to execute a single instruction repeatedly. These values are given in Table 4.3. As with a particular program or instruction mix, these values are obtained by assuming that node failures are independent and exponentially distributed.

The grouped signatures are then weighted by the fraction of instructions from that group in the hypothetical program to obtain a wide mix of program types to investigate the dependence between program type and MTTF. The results from this study are presented in the next section.

4.3 Results

The plot in Figure 4.5 shows the variation of the MTTF of the microprocessor as a

Table 4.2: Failure time (in years) of target chip for instructions

Instruction categories									
SKIP		SHIFT		REG MOD		MEM REF		IO	
Name	*MTTF*	*Name*	*MTTF*	*Name*	*MTTF*	*Name*	*MTTF*	*Name*	*MTTF*
SL	8.95	LSA	10.52	IAB	10.15	ADD	10.48	INA	10.84
SG	8.99	LLA	10.80	LSAV	10.13	SUB	11.26	OTA	10.15
SE	9.01	LSL	10.50	TCA	9.45	SAL	9.77		
SGE	8.99	LLL	10.78	OCA	9.73	MUL	13.00		
SLE	8.98	RSA	10.48	NOP	8.83	DPA	12.26		
SME	9.00	RLA	10.43	CLA	8.82	AND	9.62		
SG.I	9.07	RSL	10.45	CLB	8.81	BSA	10.47		
SE.I	9.07	RLL	10.41	CLAB	8.84	DIV	12.46		
SNO.I	9.01	LSR	10.48	AOA	8.92	STA	10.17		
SL.I	9.08	LLR	10.79	SOA	10.10	BRU	11.97		
SGE.I	9.06	RSR	10.42	LOA	8.85				
SLE.I	9.04								
SME.I	9.06								
SG.D	8.99								
SE.D	10.40								
SMO.D	10.32								
SL.D	10.40								
SGE.D	10.38								
SLE.D	10.35								
SME.D	10.38								
ADA.B	8.93								
SDA.B	10.10								
LDA.B	8.68								
SG.B	9.01								
SE.B	9.00								
SNO.B	8.96								
SL.B	8.99								
SGE.B	8.98								
SLE.B	8.99								
SME.B	9.00								
Mean=	9.31		10.55		12.97		11.14		10.53

Table 4.3: Failure times (in years) of the HS1602 for groups of instructions

Instruction categories							
SKIP		COMP				IO	
Name	*MTTF*	*Name*	*MTTF*	*Name*	*MTTF*	*Name*	*MTTF*
SL	8.95	LSA	10.52	LOA	8.85	INA	10.84
SG	8.99	LLA	10.80	DIV	12.46	OTA	10.15
SE	9.01	LSL	10.50	BRU	11.97		
SGE	8.99	LLL	10.78				
SLE	8.98	RSA	10.48				
SME	9.00	RLA	10.43				
SG.I	9.03	RSL	10.45				
SE.I	9.07	RLL	10.41				
SNO.I	9.01	LSR	10.48				
SL.I	9.08	LLR	10.79				
SGE.I	9.06	RSR	10.42				
SLE.I	9.04	IAB	10.15				
SME.I	9.06	LSAV	10.13				
SG.D	8.99	AND	9.62				
SE.D	10.40	TCA	9.45				
SMO.D	10.32	OCA	9.73				
SL.D	10.40	NOP	8.83				
SGE.D	10.38	CLA	8.82				
SLE.D	10.35	CLB	8.81				
SME.D	10.38	CLAB	8.84				
ADA.B	8.93	AOA	8.92				
SDA.B	10.10	SOA	10.10				
LDA.B	8.68	STA	10.17				
SG.B	9.01	ADD	10.48				
SE.B	9.00	SUB	11.26				
SNO.B	8.96	SAL	9.77				
SL.B	8.99	MUL	13.00				
SGE.B	8.99	DPA	12.26				
SLE.B	8.99	AND	9.62				
SME.B	9.00	BSA	10.47				
Mean=	9.31		10.55				10.53

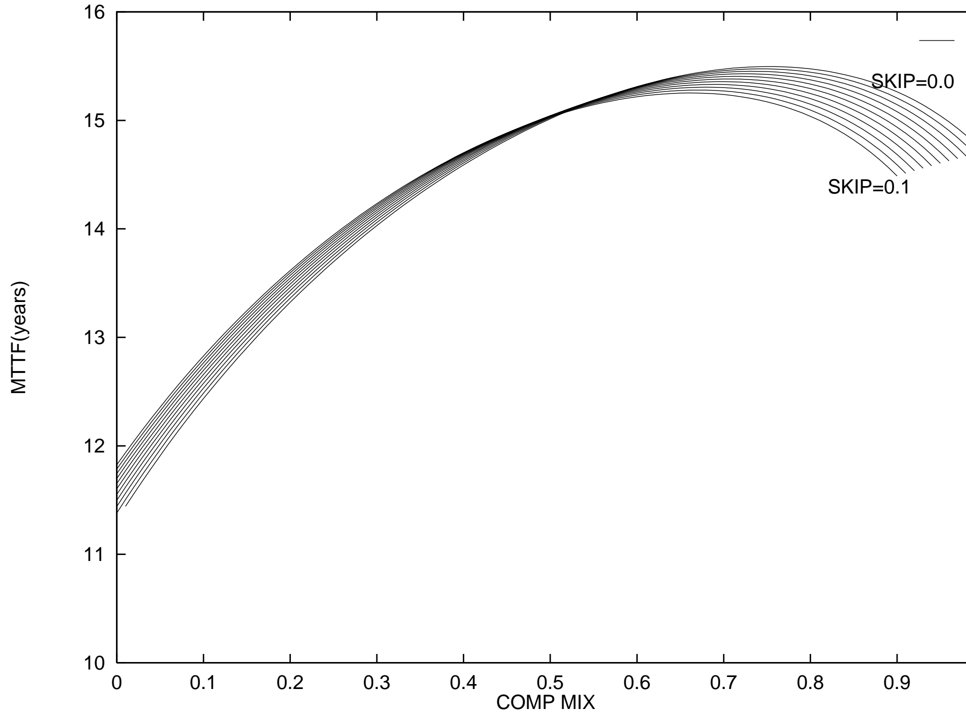

Figure 4.5: MTTF of chip with *COMP* mix[0.0,1.0] and *SKIP* mix [0.0,0.1].

function of *COMP* mix for different *SKIP* mixes. Each curve in the graph corresponds to different fractions of $SKIP$ instructions, which is varied from 0 to 0.1. The x-axis then gives, for each $SKIP$ fraction, the fraction of instructions in the mix from the $COMP$ category. The remaining fraction (1 - $SKIP$ - $COMP$) are IO instructions. We observe that as the mix of *SKIP* instructions increases from 0 to 0.1, the reliability of the microprocessor improves if the *COMP* mix is less than 50%. However, when the *COMP* instructions contribute more than half of all instructions in a given program, an increase in *SKIP* mix actually decreases the lifetime of the target chip. We also note the non-linear dependence of the chip lifetime on the instruction mixes. The reliability the chip increases initially with *COMP* mix, and after touching a maximum, it begins to

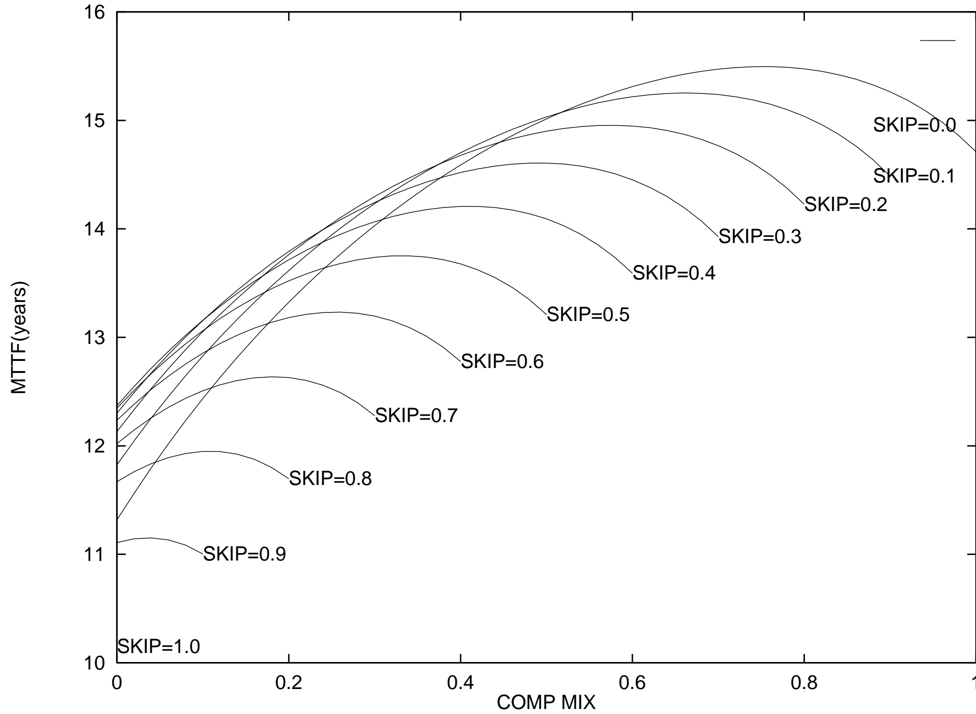

Figure 4.6: MTTF of chip with *COMP* mix[0.0,1.0] and *SKIP* mix [0.0,1.0].

decrease. The maximum MTTF occurs when the contribution of the *COMP* instructions is between .5 and .8, depending on the fraction of instructions in the $SKIP$ category.

Figure 4.6 extends the variation to a wider range of *SKIP* mixes. In this plot, *SKIP* mix is varied from 0 to 1.0 in steps of 0.1 to study broadly the impact of *SKIP* and *COMP* instructions on the dependability of the target chip. This graph makes clear the significant dependence between instruction mix and the MTTF of the microprocessor. For example, when the sample program consists of purely *IO* instructions (*COMP* mix = 0.0 and *SKIP* mix = 0.0), the MTTF of the microprocessor is about 11.31 years. A sample program with only *COMP* instructions (*SKIP* mix = 0.0 and *IO* mix = 0.0) would increase the MTTF of the microprocessor to 14.71 years. Likewise, an all-SKIP

program would lower the MTTF of the chip considerably to about 10.15 years. Mean times to failure while executing real programs, which would be a mix of these instruction types, would fall at other points on the graph. Thus the workload type does indeed significantly impact the lifetime of the target chip.

5. CONCLUSIONS AND FUTURE WORK

The goal of this study was two-fold. First, it was to propose a methodology for investigating the impact of the workload of a microprocessor on its MTTF. Second, it was to use the methodology to investigate the hypothesis that the workload affected the MTTF of the chip via study of an example microprocessor. Per the first objective, we developed a combined simulation/analysis environment that generates a signature for a program executing on the chip and uses the signature, together with an analytical electromigration model, to predict the MTTF of the program executing on the chip. The evaluation of a microprocessor executing a program is fast, since we study the effect of instructions separately, rather than executing the entire program (at the logic level) on the microprocessor, and use an analytical electromigration model. While the environment proposed uses a specific logic-level simulator and electomigration model, it can be easily changed to use other simulators and models, accounting for different assumptions about the physical parameter values of the chip being studied.

Regarding the second aim, we studied the MTTF, considering failures due to electromigration, for a wide range of mixes of skip, computation, and IO instructions. We showed that the MTTF of the HS6102 microprocessor varies significantly depending on the program that it is executing. In addition to being used for assessment, the results can also help a designer increase the dependability of a microprocessor, since the methodology pinpoints nodes that are used heavily and hence contribute most significantly to the microprocessor's failure rate. It is our hope that these results will motivate additional work regarding the impact of workload on the permanent failure rate of microprocessors.

Although the experiment described in this thesis is performed under certain specific operating conditions, the analysis could easily be extended to consider varying operating environments and fabrication technologies. Currently the SPLICE simulator supports the description of the microprocessor in *Berkeley Language Translator*, a format that is not popular with the VLSI community. Consequently, efforts are being made to design and develop a new simulator with support for more standardized formats such as VHDL and Verilog. Pipelined microprocessors pose interesting challenges in terms of data dependences and in-order execution of instructions. The application of this methodology to a generic class of pipelined microprocessors is currently under investigation. The electromigration model of the metal stripes used in this analysis has a few limitations. A more sophisticated model can be used to take into account the fringe effects and temperature dependence of the metal resistivity.

6. APPENDIX

This chapter contains sample source code for programs used in the experimental study of the HS1602 microprocessor. The software level flow of computation of MTTF values in the example study is described in Figure 6.1.

```
# Run the splice simulator on 1602.sp ( the netlist) using
#the default ROM files
#rom00.b
#rom01.b
#rom02.b
#rom03.b
# And Workload specified in ram00.b (Lower order byte) and
# ram01.b (Higher order byte)
# The output trace is collected in "tr1602"
# sun4splice is the simulator executable for sun4.
```

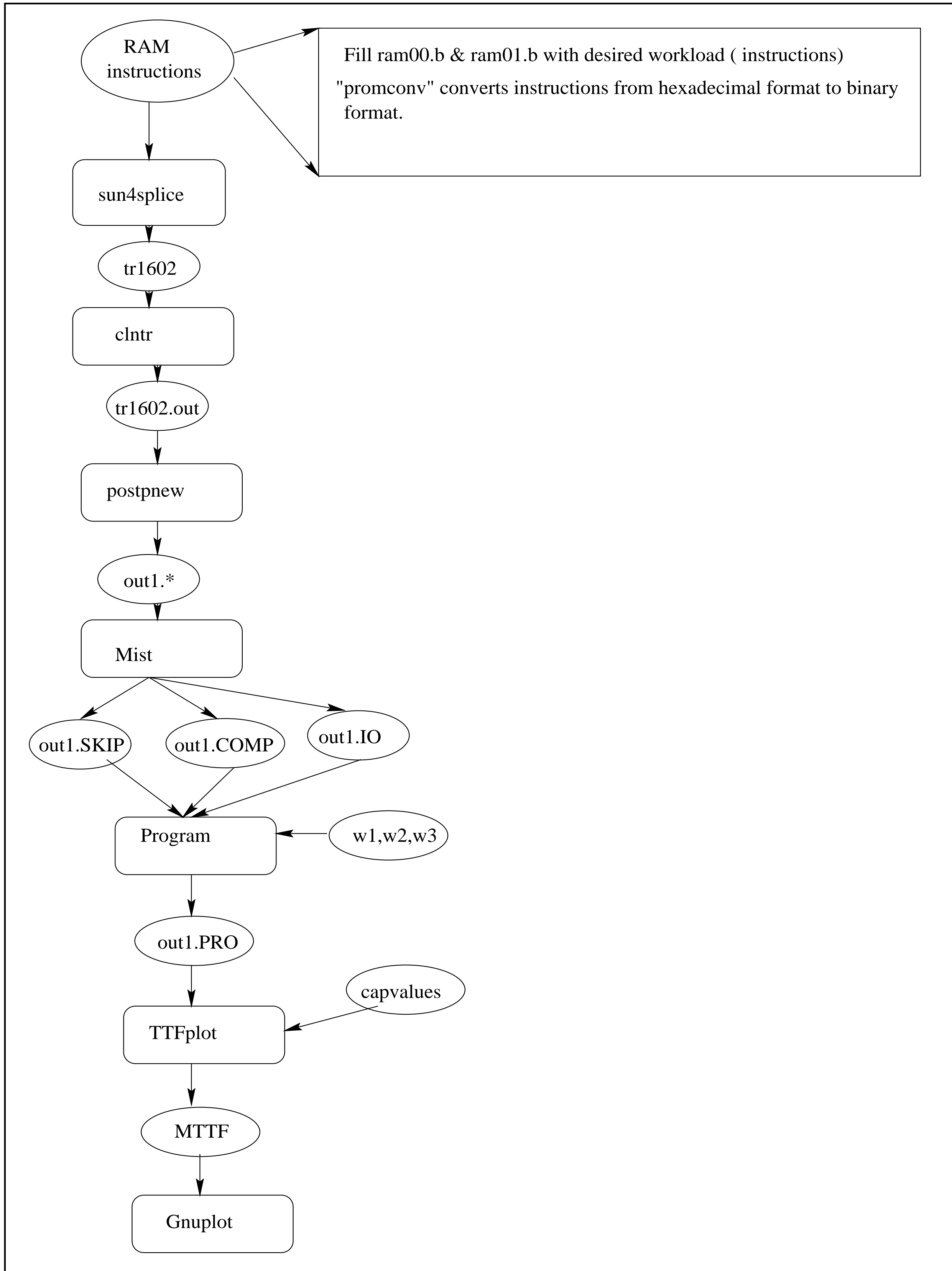

Figure 6.1: Program flow chart for computation of MTTF.

```
sun4splice -i 1602.sp -t tr1602 -o /dev/null

# Next, clean the trace to remove duplication of events.
# input : original trace file
# output : clean trace file, by default "tr1602.out"

clntr tr1602 tr1602.out
```

```
/** postpnew.c ***/
/** Postprocessor **/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NODES 3822
unsigned int Node_Value[3930];
unsigned int Node_Time[3930];
unsigned int Node_Ntimes[3930];
unsigned int Sum[3930];
```

```
main()
{

FILE  *fp, *fpout, *fpout1, *fpout2;
char f1[40],f2[30],f3[40],f4[40],f5[60];
char s[80];
int i,data,N;
unsigned int diff;
unsigned node=0;
unsigned time=0;
unsigned value=0;

fp = fopen("tr1602.out","r");
if( fp == NULL) {
perror("Could not open file");
exit(1);
}

 fpout = fopen("out","w");
if( fpout == NULL) {
perror("Could not open Output file");
exit(1);
```

```
}

 fpout1 = fopen("out1","w");
if( fpout1 == NULL) {
perror("Could not open Output1 file");
exit(1);
}

 fpout2 = fopen("out2","w");
if( fpout2 == NULL) {
perror("Could not open Output2 file");
exit(1);
}

/* dummy first line read **/
fgets(s,80,fp);
printf("\n");

/** Initialize all nodes to zero **/
for(i=0;i<3930;i++)
{

Node_Value[i]=0;
```

```
Node_Time[i]=0;

Node_Ntimes[i]=0;

Sum[i]=0;

}


while(data != EOF)

{


data=fscanf(fp, "%s %s %s %s %s", f1,f2,f3,f4,f5);

/*printf("%s %s %s %s %s\n",f1,f2,f3,f4,f5); */

/** Store logic value of node i in Node[i] **/

node=atoi(f2);

time=atoi(f1);

value=atoi(f3);

/* printf("\nTime=%d, Node=%d, val=%d",time,node,value);*/

/*

if(node == 69)

{

printf("New Value=%d\n",value);

printf("New Time=%d\n",time);

printf("Old Node Value =%d\n",Node_Value[node]);

printf("Node Ntimes =%d\n",Node_Ntimes[node]);

printf("Old Node time =%d\n",Node_Time[node]);
```

```
printf("Sum=%d\n",Sum[node]);

}

*/

diff= time - Node_Time[node];

Sum[node]=Sum[node] + diff;

Node_Ntimes[node]++;

Node_Time[node]=time;

}


for(i=1;i<3929;i++)

{

 fprintf(fpout,"%d %d\n",i,Node_Ntimes[i]);

if( (Node_Ntimes[i] == 0) || (Node_Ntimes[i]==1) || (Node_Ntimes[i]==2) )

fprintf(fpout1,"%d %f\n",i,2000.00);

/** Although Infinity, MIST is here given a high value of 2000 mtu **/

else

fprintf(fpout1,"%d %f\n",i,(double)Sum[i]/(double)Node_Ntimes[i]);

fprintf(fpout2,"%d %d\n",i,Node_Time[i]);

/*

printf("\n");

printf("\nNode Number %d switched %d times",i,Node_Ntimes[i]);

if( (Node_Ntimes[i] == 0) || (Node_Ntimes[i]==1)|| (Node_Ntimes[i]==2) )

printf("\nMean IST for Node Number %d = %f ",i,2000.0);
```

```
else
printf("\nMean IST for Node Number %d = %f ",i,(double)Sum[i]/(double)Node_Ntimes[i]);
printf("\nAnd last Switch Time = %d ",Node_Time[i]);
*/
}
fprintf(fpout,"%s %d\n",token,ntimes);
printf("\nAnd last Switch Time = %d ",last_switch);
fclose(fp);
fclose(fpout);
}


/** DOall.c ***/


#include <stdio.h>
#include <string.h>


main()
{
double c1,c2,c3;
double w1,w2,w3;
char dum[40];
unsigned int S;
```

```
w1=0.0;
w2=0.0;
w3=0.0;
for (c1=0.0;c1<=1;c1=c1+0.1)
{
w1= c1;
for(c2=0.0;c2<=1;c2=c2+0.02)
{
w2= c2 ;
for (c3=0.0;c3<=1;c3=c3+0.02)
{
w3= c3 ;
S=(unsigned int)(100*w1+100*w2+w3*100);
  if ( S == 100 )
{
sprintf(dum,"Program %f %f %f",w1,w2,w3);
system(dum);
system("TTFplot");
  }
}
}/* c2 **/
} /* c3 **/
}
```

```
/** Program.c ***/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_NODES 3928

main(argc,argv)
int argc;
char *argv[];
{
FILE  *fp1, *fp2, *fp3, *fpout;
int cont,i,data,N,j,nl;
char buf[30],dum[30];
float meen1,meen2,meen3;
double Sum=0.0;
double mix;
float w1,w2,w3;
unsigned int diff;
unsigned int l1,l2,l3;

w1=atof(argv[1]);
```

```
w2=atof(argv[2]);

w3=atof(argv[3]);

cont=0;

fp1 = fopen("out1.SKIP","r");

if( fp1 == NULL) {

perror("Could not open file");

exit(1);

}

fp2 = fopen("out1.COMP","r");

if( fp2 == NULL) {

                perror("Could not open file");

                exit(1);

                }

fp3 = fopen("out1.IO","r");

if( fp3 == NULL) {

                perror("Could not open file");

                exit(1);

                }


fpout = fopen("out1.PRO","w");

if( fpout == NULL) {

                perror("Could not open file");

                exit(1);
```

```
                }
for(N=1;N<MAX_NODES;N++)
{

fscanf(fp1,"%d %f",&nl,&meen1);
fscanf(fp2,"%d %f",&nl,&meen2);
fscanf(fp3,"%d %f",&nl,&meen3);
mix = meen1 * w1 + meen2 * w2 + meen3*w3;
fprintf(fpout,"%d %f\n",N,mix);
}
fclose(fp1);
fclose(fp2);
fclose(fp3);
fclose(fpout);
}


/*** TTFplot.c ***/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
```

```
#define MAX_NODES 3927

#define CMOSCAP 1.6e-15

#define R 6e+2

/** Define the constants **/

#define delta 1e-4      /** Grain Boundary Width ( in cm) **/

#define ZeeQ 1.6e-19    /** Positive Electron Charge **/

#define Rho 2.83e-6      /** Specific resistance of Al **/

#define D_0 1e-4        /** Intrinsic Diffucivity **/

#define thetastar  37    /** Angle of misinclination **/

#define SINTHETA 0.31730466

#define phi  60         /** Angle of Garin Boundary **/

#define N_gb  6e-10   /** Atomic Density of Al ? **/

#define alpha   4e-3   /** Coefficient of resistance for Al **/

#define Bk  1.38e-23 /** Boltzmans Constant  J/K */

#define k   8.6164 /** Boltzmans Constant  ev/K */

#define Q0  0.7 /**  eV*/

#define Duty_Factor 0.07 /**  eV*/


main()

{


double mist, dutyfactor, C;

char s[80];
```

```
FILE  *fp, *fp1, *fpout1, *fpout2;

int i,N;

int data;

int dum;

unsigned int fcap;

float meen;

unsigned int diff;

unsigned node=0;

unsigned time=0;

unsigned value=0;

char c;


/** TTF Variables **/


unsigned long t=0;

double T_a,T_tp,T_t1;

double Z0,Pd,t1,th,tp;

double j=6e+3;

double factor;

double T_Current;

double T_Prev;

double Tc=10e-9; /* initial switching instant **/

double Ti;
```

```
double S=0.0;
double ttime;
double Diffusivity;
double F_phi;
double J;
double therm_imp;
double h=3e-6; /** 300 Ang : Height **/
double V;
double Omega= 1.6e-20;
double AMU= 50e-27; /** Kg **/
double Del_Time; /** Real Time*/
double V0; /** Real Time*/
double P; /** */
double Width=5e-4; /**cm  */
double Denominator; /** */
double Ttf; /** */
double Sum=0.00;
char fname[30];
unsigned int suf,m;
char buf[20];
char b[20];
char f[40];
suf=22;
```

```
/** TTF variables end **/

T_Current = 510.234;

Sum=0.0;

fp = fopen("capvalues","r");

if( fp == NULL) {

perror("Could not open Output file");

exit(1);

}

fp1 = fopen("out1.PRO","r");

if( fp1 == NULL) {

perror("Could not open Output file");

exit(1);

}

for ( i =1; i< MAX_NODES; i++)

{

fscanf(fp,"%d",&fcap);

C= (double)fcap * CMOSCAP;

fscanf(fp1,"%d %f",&dum,&meen);

mist= meen*1e-9;

dutyfactor= (5*R*C)/( 5*R*C + mist);

/** Compute the flux for this switching !! */

/** J for this node is computed(Same as initial value ) **/

Diffusivity= D_0 * (SINTHETA) * exp( (-1* Q0)/(k * T_Current));
```

```
P= 0.887;

Denominator = Diffusivity * ZeeQ * Rho * j;

Ttf = (1000.0)*(Width * Bk * T_Current * P)/ ( Denominator );

Ttf = Ttf /( dutyfactor * dutyfactor);

Ttf= Ttf/(60.0*60.0*24.0*365.0);

Sum= Sum + (1/Ttf);

}


printf( "%0.9f ",1/Sum);

fclose(fp);

fclose(fp1);

}
```

REFERENCES

[1] J. Black, "Electromigration Failure Modes in Aluminium Metallization for Semiconductor Devices," *Proceedings of the IEEE*, vol. 57, no. 9, pp. 1587-1594, Sept. 1969.

[2] L. Brooke, "Pulsed Current Electromigration Failure Model," *IEEE Proceedings IRPS*, pp. 136-144, April 1987.

[3] G. S. Choi, "Fault-Sensitivity and Wear-Out Analysis of VLSI Systems," *Ph.D dissertation, Department of Electrical Engineering, University of Illinois at Urbana-Champaign*, July 1994.

[4] G. S. Choi and R. K. Iyer, "Wear-Out Simulation Environment for VLSI Design,"*Proceedings of FTCS-23*, pp. 320-329, June 1993.

[5] E. W. Czeck, "On the Prediction of Fault Behaviour Based on Workload," *Ph.D. dissertation, Carnegie-Mellon University*, April 1991.

[6] E. W. Czeck and D. P. Siewiorek, "Observations on the Effects of Fault Manifestation as a Function of Workload," *IEEE Transactions on Computers,* vol. 41, no. 5, pp. 559-567, May 1992.

[7] J. Dervenis, "Algorithms and Structure of SPLICE3," *Master's Thesis, University of Illinois at Urbana-Champaign*, May 1988.

[8] D. Frost and K. Poole, "RELIANT: A Reliablity Analysis Tool for VLSI Interconnects," *IEEE Journal of Solid State Circuits*, vol. 24, no. 2, pp. 458-463, April 1989.

[9] "HS1600 Design Report," *United Technologies Hamilton Standard*, April 1985.

[10] J. W. Harrison, Jr., "A Simulation Model for Electromigration in Fine-Line Metallization of Integrated Circuits Due to Repetitive Pulsed Currents," *IEEE Transactions on Electron Devices*, vol. 35, no. 12, pp. 2170-2180, Dec. 1988.

[11] F. M. d'Heurle, *Proc. IEEE*, vol. 59, 1971.

[12] C. Hu, P. Ko, P. Lee, N. Cheung, and B. Liew, "IC Reliability Prediction," *SRC TECHCON-88 Dallas, TX*, pp. 240-243, Oct. 1988.

[13] B.-K. Liew, N. W. Cheung and C. Hu, "Projecting Interconnect Electromigration Lifetime for Arbitrary Current Waveforms," *IEEE Transactions on Electron Devices*, vol. 37, no. 5, pp. 1343-1352, May 1990.

[14] P. Marcoux, P. Merchant, V. Naroditsky, and W. Rehder, "A New 2D-Simulation Model of Electromigration," *Hewlett-Packard Journal*, June 1989.

[15] D. Siewiorek and R. Swarz, "Reliable Computer Systems, Design and Evaluation," *Digital Press*, 2nd ed., 1992.

[16] R. A. Sigsbee, "Electromigration and Metalization Lifetimes," *Journal of Applied Physics*, vol. 44, no. 6, pp. 2533-2540, June 1973.

[17] D. Young and A. Christou, "Failure Mechanism Models for Electromigration," *IEEE Transactions on Reliability*, vol. 43, no. 2, pp. 186-193, June 1994.