D2CyberSoft: A Design Automation Tool for Soft Error Analysis of Dependable Cybercars

Arslan Munir
Computer Science & Engineering Department
University of Nevada, Reno, NV, USA
Email: arslan@unr.edu

Farinaz Koushanfar
Electrical & Computer Engineering Department
Rice University, Houston, TX, USA
Email: farinaz@rice.edu

***Abstract*—Next generation of automobiles (also known as cybercars) will escalate the proliferation of electronic control units (ECUs) to implement novel distributed control applications such as steer-by-wire (SBW), brake-by-wire, etc. Although the use of electronic embedded systems improves performance, driving comfort, safety, and economy for the customer; this electronic control of automotive systems makes these systems susceptible to *soft errors*, which can significantly reduce the availability of these systems. Enhancing reliability and availability at a minimum additional cost is a major challenge in automotive systems design. In this paper, we propose D2CyberSoft—a design automation tool for cybercars that facilitate cybercar designers in selecting designs with minimum cost overhead to make safety-critical automotive functions less vulnerable to soft errors. D2CyberSoft aids designers by providing built-in models, easy to specify inputs, and easy to interpret outputs. We elaborate Markov models that form the basis of D2CyberSoft using SBW as a case study. We further provide evaluation insights obtained from D2CyberSoft.**

***Keywords*—Automotive embedded systems, soft errors, Markov models, design automation, availability, dependability, x-by-wire, steer-by-wire**

I. Introduction and Motivation

Next generation of automobiles (also known as cybercars) will increase the proliferation of electronic control units (ECUs) for implementing novel automotive applications, such as steer-by-wire (SBW), brake-by-wire, throttle-by-wire (often collectively referred as x-by-wire). Although computerized control of automotive systems offers various performance, comfort, and safety benefits; this computerized control also dramatically increases the susceptibility of microprocessors to *soft errors*. Soft errors are transient errors caused by electrical noise and high energy particle strikes, such as neutrons from cosmic rays and alpha particles from the integrated circuit packaging materials. Soft errors can be pernicious to program execution by flipping the bits stored in memory or by changing the values being computed in logic elements.

Soft errors can significantly decrease the availability of automotive systems. The cyber-physical attributes of modern automotive systems directly couple automotive systems' availability to the cybercar's physical safety and dependability. Research has demonstrated that soft errors can lead to critical failures in automotive systems [1]. As automotive embedded systems permeate into safety-critical functions; safety, hazard, and availability analysis become imperative as stipulated in automotive standards. ISO 26262 is a state-of-the-art automotive standard that classifies risk by four automotive safety integrity levels (ASILs): ASIL-A, -B, -C, or -D, where ASIL-D represents the most stringent and ASIL-A the least stringent level. An ASIL specifies a function's (item's) necessary safety requirements for achieving an acceptable residual risk. To improve the availability of automotive systems and meet ASIL requirements in the presence of soft errors, fault-tolerance (FT) or dependability integration in automotive designs is imperative.

Dependability integration in automotive systems is challenging because of stringent cost constraints. In order to save wiring costs, many ECUs are moved from a somewhat isolated and comfortable place to near the actuators that are exposed to harsh environments. To protect the delicate electronics in the hostile environment of a car, each ECU is enclosed in a package. Although packaging can protect automotive electronics; radioactive contaminants in packaging materials, due to manufacturing issues, are a source of alpha particles that cause soft errors. Additionally, addressing soft errors due to neutron strikes poses a substantial problem because adequate shielding is prohibitively expensive. Dependability assimilation in automotive systems is also constrained by strict performance requirements. The integrated FT techniques to improve reliability and availability must have a minimal impact on performance because many automotive systems have hard real-time deadlines (e.g., airbag system's operation depends on ECU speed). Traditional safe states such as resetting an ECU can be a hazardous killer for availability given the increasing soft failure rates, and thus cannot be afforded for safety-critical systems.

A holistic availability analysis of automotive systems for soft errors has not been explored thoroughly in literature. Most of the contemporary reliability and availability analysis tools (e.g., ReliaSoft [2], SHARPE [3], OSATE [4]) are general-purpose and do not include built-in models for availability analysis of cybercars. Furthermore, most of prior reliability and availability software tools require a user with a high degree of expertise in reliability engineering and computer design. Developing availability models, specifying inputs to these models, and properly interpreting outputs from these tools require considerable resources and skills.

To assist designers for design space exploration and safety assessment of automotive systems, we have developed D2CyberSoft (Design of Dependable Cybercars in presence of Soft errors)—a tool that provides a systematic approach for modeling and prediction of automotive systems' availability for different soft error rates. Our main technical contributions are as follows:

- A design automation tool for design space exploration and safety (availability) assessment of dependable cybercars (D2CyberSoft) with SBW application as a case study.
- Cost-efficient cybercar design to meet various availability requirements using D2CyberSoft.
- Markov models formulation that provides basis for D2CyberSoft with SBW application as a case study.
- Comprehensive availability analysis of cybercar designs for different soft error rates.

To overcome these limitations of prior work, we develop Markov models for quantifying the availability of various FT designs. D2CyberSoft would help cybercar designers to select dependable designs with minimum cost overhead to make safety-critical automotive functions less vulnerable to soft errors.

II. Related Work

Prior work on soft errors mitigation in complementary metal-oxide-semiconductor (CMOS) has traditionally focused on a memory cells instead of combinational logic because error detection and correction schemes for memory are well known. Saleh et al. [5] analyzed transient error recovery in FT memory systems using a scrubbing technique that was based on single error correction and double error detection codes. The authors derived reliability and mean time to failure (MTTF) expressions for memory systems subjected to transient errors. Mukherjee et al. [6] studied soft errors in microprocessor considering an architecture vulnerability factor (AVF) (AVF is is the probability that a fault in a particular structure/component will result in a visible error). The authors measured the AVFs of instruction queue and execution units of an Itanium2-like IA64 processor. However, the work did not quantify the system's availability.

Li et al. [7] observed that traditional MTTF calculation methods could not give accurate MTTFs for large systems with long-running workloads and high raw error rates. The authors suggested that an alternative method, SoftArch, provided better MTTF estimates for such systems. The work, however, focused only on MTTF estimation for high-end servers and did not consider MTTF or availability assessment of embedded processors in the presence of soft errors.

Soft error analysis of embedded systems has been done in previous work. Blome et al. [8] analyzed the effect of soft errors on an ARM926EJ-S embedded processor. Ossi [9] studied architecture-level soft error detection and correction strategies that targeted the arithmetic logic units (ALU) within a microprocessor. Wang [10] developed an analytical model for neutron-induced soft error rate (SER) estimation and calculated the SERs for ISCAS85 benchmark circuits. However, the model required further validation as the SER estimation results from the developed model depicted significant differences from previous SER estimation works [11][12]. We point out that the SER estimation works are complementary to our work, as D2CyberSoft requires ECUs' SER estimation.

Some prior work explored permanent and transient faults in automotive systems. [13] elucidated the modeling of permanent faults in automotive systems. Skarin et al. [1] studied the impact of soft errors in a prototype brake-by-wire system. Results revealed that 30% of the injected errors (bit flips) passed undetected and caused the ECU to produce erroneous outputs for the brake actuator, 15% of which resulted in critical failures. However, the work focused on error detection and recovery and did not quantify the availability of the brake-by-wire system in the presence of soft errors.

Several earlier works elaborated dependability related tools and methodologies. Johnson et al. [14] discussed the purpose and type of models used for various reliability, availability, and serviceability modeling tools. Lambert [15] discussed the use of fault tree analysis to assess failure modes within automotive systems using a car starting system as a case study. The work only conducted a qualitative evaluation and did not present quantitative (probabilistic) insights on the effects of failure in automotive systems.

III. D2CyberSoft

D2CyberSoft is a design automation tool for cybercars that facilitate designers in selecting designs with minimum cost overhead to make safety-critical automotive functions less vulnerable to soft errors. This section elucidates optimization objective function, inputs and outputs, and design methodology for D2CyberSoft.

A. Optimization Objective Function

D2CyberSoft solves an optimization problem to determine a cost-effective cybercar design under various constraints. The optimization problem can be formulated as:

$$\begin{array}{lll} \min & cost_d & \\ s.t. & \mathcal{A} \geq a, & a \in \mathbb{R}_{[0,1]} \\ & e_{c_i} = b_i, & \forall\ c_i \in S,\ b_i \in \mathbb{R}_{\geq 0} \\ & N_{c_i} = n_i, & \forall\ c_i \in S,\ n_i \in \mathbb{N} \\ & \text{ASIL}_S \geq y, & y \in \{\text{A,B,C,D}\} \\ & \mathcal{L} = t, & t \in \mathbb{R}_{\geq 0} \end{array} \quad (1)$$

where $cost_d$ denotes design cost, which depends on the level of redundancy integrated in the design for improving dependability. $\mathcal{A}$ denotes target availability of automotive system S to be designed (e.g., SBW), e_{c_i} denotes raw SER of the component i in failure in time (FIT) per bit (1 FIT = $1{\times}10^{-9}$ failures per hour) and N_{c_i} denotes total number of bits in the component c_i of the automotive system S. D2CyberSoft permits the designer to specify components' size (e.g., cache size) from which the tool then estimates the SER of the system. ASIL$_S$ specifies the ASIL requirement of the design and $\mathcal{L}$ denotes the desired lifetime for the designed automotive system (for warranty period specification). We point out that if any of the input parameters are missing (e.g., e_{c_i} or N_{c_i} of a component), then default values of these parameters specified in D2CyberSoft is used. $\mathbb{R}_{\geq 0}$ denotes the set of real numbers greater than or equal to zero and $\mathbb{N}$ denotes the set of natural numbers.

B. D2CyberSoft Inputs, Outputs, & Design Methodology

D2CyberSoft takes input from the designer on the desired availability, ASIL, lifetime, components' number and size, and outputs a minimum cost design that satisfies the design constraints. Availability can be specified in terms of number of nines (e.g., 0.999 or 99.9% has three nines). D2CyberSoft provides built-in Markov models for cybercar design focusing on a SBW application (Section V). D2Cyber Markov models are specified in SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator), which requires soft failure rate (SFR) specification of ECUs involved in the design.

D2CyberSoft calculates the SFR of a processor in two steps [7]: (1) AVF step—the raw SER of each individual processor component (e.g., ALU, register file, etc.) is multiplied by an AVF factor to determine the SFR of that component; and (2) sum of failure rates (SOFR) step—the SFRs of the individual components of the processor are summed to calculate the entire processor's SFR. The above SFR calculation procedure assumes that the probability of failure due to a soft error in a given component is uniform across a program's execution, which allows a single AVF value to be used to derate the raw SER of the component. This uniformity assumption is reasonable for raw soft error events since the probability of high energy particle strikes is no different at different points in the program's execution for most realistic scenarios. The SOFR step assumes that the time to failure for a given component follows an exponential distribution and the failures for different components are independent of each other. This exponential distribution assumption is reasonable because the time to the next high energy particle strike is independent of the previous strike. The errors introduced due to AVF and SOFR assumptions are negligible for our SBW case study where the total number of components is small and the workload consists of repeated executions of a short program [7].

The SFR of a processor component c_i in D2CyberSoft is determined as:

$$\lambda_{s\_c_i} = (N_{c_i} \cdot e_{c_i}) \times \mathcal{F}_c \tag{2}$$

where N_{c_i} denotes total number of bits in the component c_i, e_{c_i} denotes raw SER of the component c_i, and $\mathcal{F}_{c_i}$ denotes AVF of the component c_i.

For an automotive ECU with main components as instruction cache, data cache, random access memory (RAM), register file, instruction fetch/decode unit (IFU), integer execution unit (IEU), and floating point unit (FPU); D2CyberSoft estimates the ECU's overall SFR λ_{s_ECU} as:

$$\begin{aligned}\lambda_{s\_ECU} &= \lambda_{s\_I-SRAM} + \lambda_{s\_D-SRAM} + \lambda_{s\_RAM} \\ &\quad + \lambda_{s\_reg} + \lambda_{s\_IFU} + \lambda_{s\_IEU} + \lambda_{s\_FPU}\end{aligned} \tag{3}$$

where λ_{s_I-SRAM}, λ_{s_D-SRAM}, λ_{s_RAM}, λ_{s_reg}, λ_{s_IFU}, λ_{s_IEU}, and λ_{s_FPU} denote SFR of the instruction cache, data cache, RAM, register file, IFU, IEU, and FPU, respectively. D2CyberSoft uses this ECU's overall SFR in the developed Markov models to determine a design's availability. D2CyberSoft leverages a greedy approach to solve the optimization problem for design space exploration (Eq. 1). D2CyberSoft explores the design space starting from the lowest cost design (normally the design with least redundancy) and determines the design's availability. D2CyberSoft returns the design configuration as soon as the availability of the explored design meets the specified requirements.

IV. FT Designs in D2CyberSoft

Automotive systems are subjected to stringent cost constraints and hence the additional cost for dependability integration must be minimized by introducing only as much robustness as needed and not more. The added redundancy in an FT design should have a minimal impact on automotive system's performance as these systems have hard real-time deadlines. We restrict our dependability study to techniques that leverage only dual modular redundancy (DMR) and triple modular redundancy (TMR) considering that stringent cost constraints in automotive would make the incorporation of higher N-modular redundancies (i.e., $N \geq 4$) infeasible. This section discusses a few FT designs whose built-in models are provided in D2CyberSoft.

Acceptance Tests: In contrast to permanent faults, which make the faulty component useless, soft errors only induce transient faults that can detected by error detection techniques and can be recovered by re-computations. "Acceptance tests" or "range tests" test the acceptability of outputs or results, and are the only means for error detection when only a single ECU processor core is operational. D2CyberSoft implementation of acceptance tests incorporates comparisons with expected pre-computed ranges of signals, reasonable range of changes to the current signals, and the computations execution time comparison with estimated execution times. We point out that the error detection capabilities of these acceptance tests are limited. Hence, considering the inherent fallibility of acceptance tests, D2CyberSoft assigns a large penalty (i.e., hazard penalty t_h) for error recovery that is dependent only on acceptance tests. D2CyberSoft defines the time to detect an error from acceptance tests t_d^a as:

$$t_d^a = p_s \cdot t_s + p_f \cdot t_h \tag{4}$$

where p_s denotes the probability of successfully detecting an error through acceptance tests and p_f denotes the probability of failure to detect an error through acceptance tests. t_s denotes the time to run acceptance tests t_a and the time to run the complete computation t_c in a given FT configuration, i.e., $t_s = t_a + t_c$. t^h in Eq. 4 denotes the hazard penalty, i.e., penalty associated with inability to detect an error.

D2CyberSoft designates t_h as a multiple of t^s (t^s denotes the time to run the acceptance test t_a and complete computation t_c in a given FT configuration). D2CyberSoft assigns hazard penalty values depending on the ASIL specified in design requirements (Eq. 1) to reflect the severity of the hazard associated with the error detection inability, i.e., smallest hazard penalty for ASIL-A and greatest hazard penalty for ASIL-D.

Non-fault-tolerant (NFT) design: An NFT design does not leverage any redundancy to improve reliability and availability. The NFT design cannot detect any permanent fault (hardware failure) or transient fault by itself. In an NFT design, acceptance tests provide only means to detect an erroneous computation caused by a transient fault.

FT by redundant multi-threading (FT-RMT): FT-RMT leverages dual modular redundancy (DMR) such that the (safety-critical) computation is performed on two redundant threads. The output of the threads is compared at the end of the computation. If the output of the threads match, no error occurs in the computation otherwise the computation is erroneous and needs re-computation. FT-RMT can detect one permanent or transient fault but is unable to correct errors from the two threads' outputs. Re-computation is required to correct a transient fault in an FT-RMT design. FT-RMT introduces some performance overhead as compared to an NFT design due to inherent parallelism overhead to manage threads as well as the cost of additional comparison instructions to compare the threads' outputs.

FT-RMT enhanced with quick error detection (FT-RMT-QED): FT-RMT-QED leverages DMR and further enhances FT-RMT for quick error detection (QED) [16]. FT-RMT-QED inserts additional comparison instructions at different points in the computation to detect errors early in the computation. In case of error detection at a check point in FT-RMT-QED, erroneous computation is aborted and re-computation is performed immediately to obtain an error-free output. This early error detection and correction (by re-computation) can be propitious for safety-critical applications with hard real-time deadlines.

FT-RMT with triple modular redundancy (FT-RMT-TMR): FT-RMT-TMR uses triple modular redundancy (TMR) such that the safety-critical computation is performed on three threads. A majority voter compares the outputs of the three threads and selects the majority voted output. FT-RMT-TMR can detect two permanent or transient faults, and can correct one permanent or transient fault from the threads' outputs. The two transient errors can be corrected by re-computation in FT-RMT-TMR.

FT-RMT-TMR enhanced with quick error detection (FT-RMT-TMR-QED): FT-RMT-TMR-QED leverages TMR and further enhances FT-RMT-TMR for QED. FT-RMT-TMR-QED introduces majority voting at different points in the program to detect and correct errors early in the computation. A single error is corrected by majority voting at a check point in the program whereas the technique allows early re-computation in case of two errors detection at the check point.

V. Markov Models Formulation for Soft Errors

In this section, we discuss Markov models formulation for soft errors, which forms the basis of D2CyberSoft. Improving reliability and availability by separate redundant ECUs in automotives is less feasible from wiring, space, and cost perspective. Fortunately, multi-cores provide a promising solution to incorporate redundancy at a minimum additional cost and hence we analyze the availability of multi-core-based designs with SBW application as a case study. We point out that D2CyberSoft is equally valid for designs leveraging separate redundant ECUs.

A. Steer-by-Wire System

An SBW system replaces a mechanical steering system with ECUs, sensors, and actuators, which interact via a communication bus, such as controller area network (CAN) or FlexRay. An SBW system provides various advantages over mechanical steering systems. For example, an SBW system eliminates the risk of steering column entering the cockpit in the event of a frontal crash. Since steering column is one of the heaviest components in the vehicle, removing the steering column reduces the vehicle's weight and therefore lessens fuel consumption.

Our SBW case study architecture leverages multi-core ECUs to assimilate dependability. The architecture consists of two multi-core (dual-core or triple-core) hand wheel ECUs (HW ECU1 and HW ECU2) and two multi-core (dual-core or triple-core) front axle actuator ECUs (FAA ECU1 and FAA ECU2). Each of the ECUs is connected to CAN. Our SBW architecture consists of three hand wheel sensors (hws1, hws2, and hws3) that are placed near the hand wheel to measure the driver's requests in terms of hand wheel angle, hand wheel torque, and hand wheel speed. Similarly, three front axle sensors (fas1, fas2, and fas3) measure the front axle position. Both the hand wheel sensors (the front axle sensors) are connected to the HW ECUs (FAA ECUs) by point-to-point links. Two front axle actuator (FAA) motors (FAA motor 1 and FAA motor 2) operate in active redundancy on the front axle while two hand wheel (HW) motors (HW motor 1 and HW motor 2) operate in active redundancy on the hand wheel.

An SBW system aims to provide two main services [17]: 1) front axle control (FAC) that controls the wheel direction in accordance with the driver's request, and 2) hand wheel force feedback (HWF) that provides a mechanical-like force feedback to the hand wheel. In our SBW architecture, the FAC function's implementation utilizes HW ECU1 and FAA ECU1 whereas the HWF function's implementation utilizes HW ECU2 and FAA ECU2.

B. Markov Models

Cybercar systems can be modeled by Markov chains consisting of various states. When a failure occurs in a given state, the system's next operating state is determined completely by its current state regardless of how the system entered the current state. For illustration purposes, this section discusses Markov modeling formulation of soft errors for an SBW system as a case study.

Fig. 1 depicts Markov model for transient faults (soft errors) in an SBW system with triple-core HW ECU1 and FAA ECU1. The system can detect two soft errors and correct one soft error if majority voting is performed at the end of computation as in FT-RMT-TMR. Each state (i, j) represents the state when i processor cores of HW ECU1 and j processor cores of FAA ECU1 are functioning without incurring any soft error. The solid arrows represent state changes due to soft errors in HW ECU1 and FAA ECU1 processor cores and dotted arrows represent recovery from soft errors. The initial state is $(3, 3)$ in which the three processor cores of both HW ECU1 and FAA ECU1 are functioning without incurring any soft error.

In Fig. 1, λ_{s_he} and λ_{s_fe} denote the failure rates of HW ECU1 and FAA ECU1 due to soft errors where the failure times due to soft errors are exponentially distributed. μ_{xhe} and μ_{xfe} where $x \in \{0, 1, 2\}$ denote the soft error recovery rates of HW ECU1 and FAA ECU1, respectively, as described in detail below. The differential equations describing this Markov

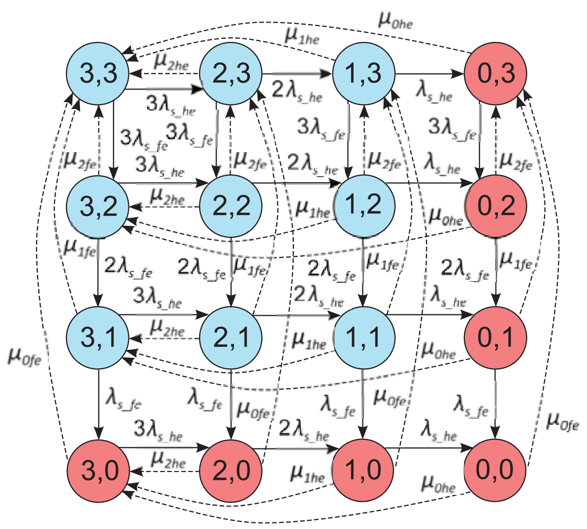

Fig. 1: Markov model for soft errors in an SBW system with triple-core HW ECU1 and FAA ECU1.

model are given as:

$$\begin{aligned}P'_{(3,3)}(t) &= -3\lambda_{s\_he}P_{(3,3)}(t) - 3\lambda_{s\_fe}P_{(3,3)}(t)\\ &\quad +\mu_{2he}P_{(2,3)}(t) + \mu_{1he}P_{(1,3)}(t)\\ &\quad +\mu_{0he}P_{(0,3)}(t) + \mu_{2fe}P_{(3,2)}(t)\\ &\quad +\mu_{1fe}P_{(3,1)}(t) + \mu_{0fe}P_{(3,0)}(t)\\ &\quad\vdots\\ P'_{(0,0)}(t) &= \lambda_{s\_he}P_{(1,0)}(t) + \lambda_{s\_fe}P_{(0,1)}(t)\\ &\quad -\mu_{0he}P_{(0,0)}(t) - \mu_{0fe}P_{(0,0)}(t) \qquad (5)\end{aligned}$$

Since an SBW system can recover from soft errors during operation, Markov models for transient faults include recovery rates. We point out that $\mu_{xhe} = \mu_{xhe}^{TMR}$ and $\mu_{xfe} = \mu_{xfe}^{TMR}$ in Fig. 1 and Eq. 5 where $x \in \{0,1,2\}$ (recovery rates are denoted as μ_{xhe} and μ_{xfe} for conciseness). The soft error recovery rates for TMR configuration (Fig. 1) are given by the following equations:

$$\mu_{2he}^{TMR} = 1/(t_{d^1_{HW}}^{TMR} + t_{c^1_{HW}}^{TMR}) \qquad (6)$$

where μ_{2he}^{TMR} denotes the soft error recovery rate of HW ECU1 from states $(2,3)$, $(2,2)$, $(2,1)$, and $(2,0)$; $t_{d^1_{HW}}^{TMR}$ denotes the time to detect a single soft error in TMR configuration for HW ECU1 and $t_{c^1_{HW}}^{TMR}$ denotes the time to correct a single soft error in TMR configuration for HW ECU1. We point out that the error detection time in any of our implemented FT configuration in D2CyberSoft (Section IV) includes the time to run acceptance tests to further verify the computation's correctness. For states (i,j) $s.t.$ $i,j \geq 2$, FT mechanisms (e.g., comparison of outputs from redundant threads) exist in addition to acceptance tests for soft error detection whereas for states (i,j) $s.t.$ $i,j \leq 1$, acceptance tests are the only means for error detection. Similarly,

$$\mu_{1he}^{TMR} = 1/(t_{d^2_{HW}}^{TMR} + t_{c^2_{HW}}^{TMR}) \qquad (7)$$

where μ_{1he}^{TMR} denotes the soft error recovery rate of HW ECU1 from states $(1,3)$, $(1,2)$, $(1,1)$, and $(0,1)$; $t_{d^2_{HW}}^{TMR}$ denotes the time to detect two soft errors in TMR configuration and $t_{c^2_{HW}}^{TMR}$ denotes the time to correct two soft errors in TMR configuration for HW ECU1. We point out that two soft errors are corrected by rerunning the program in TMR configuration and determining the correct output through majority voting. Similarly,

$$\mu_{0he}^{TMR} = 1/(t_{d^3_{HW}}^{TMR} + t_{c^3_{HW}}^{TMR}) \qquad (8)$$

where μ_{0he}^{TMR} denotes the soft error recovery rate of HW ECU1 from states $(0,3)$, $(0,2)$, $(0,1)$, and $(0,0)$; $t_{d^3_{HW}}^{TMR}$ denotes the time to detect three soft errors in TMR configuration and $t_{c^3_{HW}}^{TMR}$ denotes the time to correct three soft errors in TMR configuration for HW ECU1. The three soft errors are detected by comparisons and acceptance tests, and are corrected by rerunning the program in TMR configuration and determining the correct output through majority voting. The soft error recovery rate equations for FAA ECU1 can be written similarly.

Solving differential equations (Eq. 5) for the Markov model depicted in Fig. 1 yields solution for $P_{(i,j)}(t), \forall\, i,j \in \{0,1,2,3\}$. The (point) availability, which is the probability that the system is operational at time t, of the FAC function of the SBW system with triple-core HW ECU1 and FAA ECU1, $A_{SBW\_FAC}^{TMR}(t)$, is given by:

$$\begin{aligned}A_{SBW_{FAC}}^{TMR}(t) &= P_{(3,3)}(t) + P_{(2,3)}(t) + P_{(1,3)}(t)\\ &\quad + P_{(3,2)}(t) + P_{(2,2)}(t) + P_{(1,2)}(t)\\ &\quad + P_{(3,1)}(t) + P_{(2,1)}(t) + P_{(1,1)}(t) \qquad (9)\end{aligned}$$

Fig. 2 depicts Markov model for transient faults in an SBW system with dual-core HW ECU1 and FAA ECU1. The differential equations describing this Markov model are:

$$\begin{aligned}P'_{(2,2)}(t) &= -2\lambda_{s\_he}P_{(2,2)}(t) - 2\lambda_{s\_fe}P_{(2,2)}(t)\\ &\quad +\mu_{1he}P_{(1,2)}(t) + \mu_{0he}P_{(0,2)}(t)\\ &\quad +\mu_{1fe}P_{(2,1)}(t) + \mu_{0fe}P_{(2,0)}(t)\\ &\quad\vdots\\ P'_{(0,0)}(t) &= \lambda_{s\_he}P_{(1,0)}(t) + \lambda_{s\_fe}P_{(0,1)}(t)\\ &\quad -\mu_{0he}P_{(0,0)}(t) - \mu_{0fe}P_{(0,0)}(t) \qquad (10)\end{aligned}$$

We point out that $\mu_{xhe} = \mu_{xhe}^{DMR}$ and $\mu_{xfe} = \mu_{xfe}^{DMR}$ in Fig. 2 and Eq. 10 where $x \in \{0,1\}$ (recovery rates are denoted as μ_{xhe} and μ_{xfe} for conciseness). The soft error recovery rates for DMR configuration (Fig. 2) are given by the following equations:

$$\mu_{1he}^{DMR} = 1/(t_{d^1_{HW}}^{DMR} + t_{c^1_{HW}}^{DMR}) \qquad (11)$$

where μ_{1he}^{DMR} denotes the soft error recovery rate of HW ECU1 from states $(1,2)$, $(1,1)$, and $(0,1)$; $t_{d^1_{HW}}^{DMR}$ denotes the time to detect a single soft error in DMR configuration for HW ECU1 and $t_{c^1_{HW}}^{DMR}$ denotes the time to correct a single soft error in DMR configuration for HW ECU1. Similarly,

$$\mu_{0he}^{DMR} = 1/(t_{d^2_{HW}}^{DMR} + t_{c^2_{HW}}^{DMR}) \qquad (12)$$

where μ_{0he}^{DMR} denotes the soft error recovery rate of HW ECU1 from states $(0,2)$ and $(0,1)$; $t_{d^2_{HW}}^{TMR}$ denotes the time to detect two soft errors in DMR configuration for HW ECU1 and $t_{c^2_{HW}}^{DMR}$ denotes the time to correct two soft errors in DMR

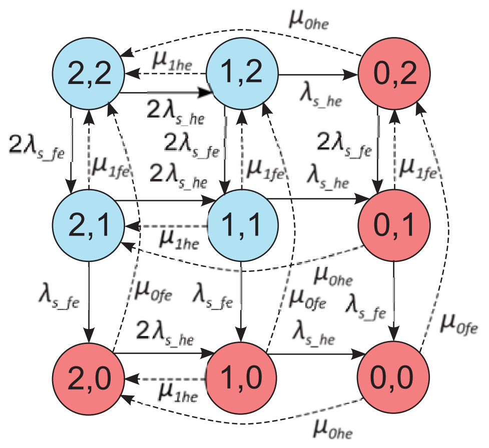

Fig. 2: Markov model for soft errors in an SBW system with dual-core HW ECU1 and FAA ECU1.

configuration for HW ECU1. The soft error recovery rate equations for FAA ECU1 can be written similarly.

Solving Eq. 10 with initial conditions $P_{(2,2)}(0) = 1$, $P_{(1,2)}(0) = 0$, $P_{(2,1)}(0) = 0$, $P_{(1,1)}(0) = 0$, $P_{(0,2)}(0) = 0$, $P_{(0,1)}(0) = 0$, $P_{(2,0)}(0) = 0$, and $P_{(1,0)}(0) = 0$ yields the solution for $P_{(i,j)}(t), \forall\, i,j \in \{0,1,2\}$. The point availability of the FAC function of the SBW system with dual-core ECUs, $A^{DMR}_{SBW\_FAC}(t)$, is given by:

$$A^{DMR}_{SBW_{FAC}}(t) = P_{(2,2)}(t) + P_{(1,2)}(t) + P_{(2,1)}(t) + P_{(1,1)}(t) \qquad (13)$$

Markov model for permanent faults in an SBW system with single-core HW ECU1 and FAA ECU1 is a subset of Fig. 2 with (1,1), (0,1), (1,0), and (0,0) as the only valid states. The point availability of the FAC function of the SBW system with single-core ECUs, $A_{SBW\_FAC}(t)$, is given by:

$$A_{SBW_{FAC}}(t) = P_{(1,1)}(t) \qquad (14)$$

Markov models for other implemented FT designs in D2CyberSoft (Section IV) can be developed similarly.

VI. Evaluation Results & Insights

In this section, we present availability evaluation results and insights obtained from D2CyberSoft. In our evaluations, we calculate repair rates (e.g., Eq. 6) corresponding to error detection and correction times for security related computations (i.e., SHA-2-based message authentication and AES-128 encryption) in cybercars. The execution times are estimated for a 32-bit embedded processor operating at 200 MHz (e.g., Freescale's MPC5746M [18]) from the measured clock cycles. We obtain these timings for security primitives because both security and dependability are imperative for cybercar design. We note that some aberrations in the presented results are due to slight inaccuracies in SHARPE calculations.

ECU's Soft Failure Rate (SFR) Calculation: For our Markov modeling analysis, we need to calculate ECUs' SFR. We model Freescale's MPC5746M ECU [18] as closely as possible for SFR calculation. The MPC5746M ECU consists of e200z4 computational cores that can operate up to 200 MHz [19][18]. The MPC5746M ECU consists of two IEUs, one IFU, one FPU, 32 general purpose registers and 110 additional registers, 8 KB instruction cache, 4 KB data cache, 16 KB instruction RAM, and 64 KB data RAM [19][18].

We assign the raw soft error rate of different ECU components as [20]: SRAM = 1×10^{6} FIT per bit, DRAM = 1×10^{1} FIT per bit, and register file = 1×10^{5} FIT per register. The raw SERs of IFU, IEU, and FPU are set to be (1/30)x, (1/10)x, and (1/10)x, respectively, of the raw SER of SRAM [20]. The raw SFR of a component is calculated by multiplying the raw SER of the component with the AVF for that component. We assign the AVFs for different ECU components as [6][21]: level one (L1) data bits = 0.18, L1 tag bits = 0.23, register file = 0.23, IFU = 0.14, IEU = 0.04, and FPU = 0.04. The MPC5746M ECU's overall SFR is calculated by the summation of SFRs of ECU's components [7] (calculations are not shown here for brevity).

Effect of FT Design on Availability: Design of an automotive system has a significant impact on the system's availability. Table I presents point availability evaluations for different SBW designs obtained form our Markov models when hazard penalty $t_h = 10^{20} \cdot t_s$ and $\lambda_{s\_he} = \lambda_{s\_fe} = 2.088167 \times 10^{-4}$/h. Results reveal that the point availability of SBW designs that leverage FT techniques is considerably higher than that of an NFT design at all time instants. For example, SBW designs leveraging either of FT-RMT, FT-RMT-QED, FT-RMT-TMR, and FT-RMT-TMR-QED provide 39x higher availability on average than an NFT design when t = 8,760 hours and $\lambda_{s\_he} = \lambda_{s\_fe} = 2.088167 \times 10^{-4}$/h. Results also show that FT-RMT-QED enables 0.00054% higher availability than FT-RMT on average. The impact of FT designs on availability for various SFRs is further discussed below.

Effect of Elapsed Time on Availability: Point availability of a system depends on the point of time the availability is evaluated. Results indicate that availability decreases as the elapsed time increases. For example, the point availability at t = 87,600 hours is 29.2% less than the point availability at t = 26,280 hours for an NFT SBW system when $\lambda_{s\_he} = \lambda_{s\_fe} = 2.088167 \times 10^{-6}$/h. Results reveal that the effect of elapsed time on point availability is more severe for an NFT design as compared to the designs with FT. For example, the point availability decreases by 11.6% for an NFT SBW design whereas the point availability decreases by 3.74×10^{-5}%, 8.63×10^{-4}%, and 3.55×10^{-4}% for FT-RMT, FT-RMT-QED, and FT-RMT-TMR, respectively, as the time elapses from t = 17,520 hours to t = 43,800 hours when $\lambda_{s\_he} = \lambda_{s\_fe} = 2.088167 \times 10^{-6}$/h.

Effect of FT Designs on Availability for Various Soft Failure Rates: SFRs have a significant impact on the availability attainable by various designs. Table II presents point availability evaluations obtained from our Markov models for NFT and FT designs for different SFRs at t = 17,520 hours $\approx$ 2 years when hazard penalty $t_h = 10^{20} \cdot t_s$. Results reveal that the designs leveraging FT techniques enable higher availability than an NFT design for all SFRs. For example, an SBW design leveraging FT-RMT, FT-RMT-QED, FT-RMT-TMR, and FT-RMT-TMR-QED provide 150,457%, 150,459%, 150,458%, and 150,459% higher availability than an NFT design when t = 17,520 hours and ECUs' SFR is

TABLE I: Point availability for various SBW designs when hazard penalty $t_h = 10^{20} \cdot t_s$ and SFR = 2.088167×10^{-4}/h.

t (hours)	NFT	FT-RMT	FT-RMT-QED	FT-RMT-TMR	FT-RMT-TMR-QED
100 h	0.959,096,733,530	0.999,999,926,600	0.999,999,988,340	0.999,999,950,100	1.000,000
4,380 h = 6 months	0.160,536,444,650	0.999,996,781,630	0.999,999,489,220	0.999,997,814,370	1.000,000
8,760 h = 1 year	0.025,771,950,062	0.999,993,563,180	0.999,998,978,450	0.999,995,628,750	1.000,000
17,520 h = 2 years	6.6419×10^{-4}	0.999,987,126,300	0.999,997,956,900	0.999,991,257,51	1.000,000
35,040 h = 4 years	4.4115×10^{-7}	0.999,974,252,610	0.999,995,913,800	0.999,982,515,070	1.000,000
52,560 h = 6 years	2.9300×10^{-10}	0.999,961,379,010	0.999,993,870,700	0.999,973,772,700	1.000,000

TABLE II: Point availability for SBW designs at t = 17,520 hours for different SFRs when hazard penalty $t_h = 10^{20} \cdot t_s$.

SFR	NFT	FT-RMT	FT-RMT-QED	FT-RMT-TMR	FT-RMT-TMR-QED
2.088167×10^{-8}/h	0.999,268,571,100	0.999,995,238,190	1.000,000	1.000,000	1.000,000
2.088167×10^{-4}/h	6.6419×10^{-4}	0.999,987,126,300	0.999,997,956,900	0.999,991,257,510	1.000,000
2.088167×10^{-1}/h	0.000,000	0.998,023,281,050	0.998,422,498,320	0.999,991,574,490	0.999,642,502,450
2.088167×10^{1}/h	0.000,000	2.5325×10^{-9}	1.4283×10^{-7}	0.999,358,712,300	0.999,219,903,940
2.088167×10^{2}/h	0.000,000	0.000,000	0.000,000	0.528,522,849,330	0.458,452,777,070

TABLE III: Steady-State availability for SBW designs for different SFRs when hazard penalty $t_h = 10^{6} \cdot t_s$.

SFR	Without FT	FT-RMT	FT-RMT-QED	FT-RMT-TMR	FT-RMT-TMR-QED
2.088167×10^{-1}/h	0.949,890,935,910	0.999,999,981,010	0.999,999,984,260	0.999,999,998,810	0.999,999,998,120
2.088167×10^{0}/h	0.629,517,133,050	0.999,998,112,670	0.999,998,437,710	0.999,999,985,070	0.999,999,939,430
2.088167×10^{1}/h	7.7014×10^{-2}	0.999,811,639,610	0.999,844,888,430	0.999,999,178,750	0.999,998,869,670
2.088167×10^{2}/h	1.3684×10^{-3}	0.981,491,294,560	0.984,713,859,350	0.999,989,021,290	0.999,978,608,980
2.088167×10^{3}/h	1.4643×10^{-5}	0.267,136,035,390	0.319,139,704,600	0.992,647,515,820	0.987,875,687,160

2.088167×10^{-8}/h. Results divulge that FT-RMT-QED enables 0.0011% and 0.04% higher availability than FT-RMT when ECUs' SFRs are 2.088167×10^{-4}/h and 2.088167×10^{-1}/h, respectively. Results show that FT-RMT-TMR enables higher availability than FT-RMT and FT-RMT-QED for various SFRs. For example, FT-RMT-TMR enables 22% higher availability than FT-RMT and 17% higher availability than FT-RMT-QED when t = 17,520 hours and SFR is 2.088167/h.

Results reveal that sophisticated FT techniques become imperative for designs that are to be deployed in harsh environments susceptible to high SERs. Results in Table II indicate that as SFR increases (i.e., 2.088167×10^{1}/h and 2.088167×10^{2}/h), only designs that leverage FT-RMT-TMR and FT-RMT-TMR-QED provide reasonable availability whereas availability furnished by other designs including FT-RMT and FT-RMT-QED approaches zero. Interestingly FT-RMT-TMR-QED is not able to provide higher availability than FT-RMT-TMR on many occasions due to additional overhead associated with the former. For example, FT-RMT-TMR enables 0.035%, 0.014%, and 15.3% higher availability than FT-RMT-TMR-QED when ECUs' SFRs are 2.088167×10^{-1}/h, 2.088167×10^{1}/h, and 2.088167×10^{2}/h, respectively, and t = 17,520 hours.

Effect of Hazard Penalty on Availability: Hazard penalty, which quantifies the penalty associated with the acceptance tests' failure for error detection, impacts availability. Results reveal that availability increases in general for all FT designs as the hazard penalty decreases, however, this availability increase is more conspicuous for designs with no or less FT. Furthermore, this availability increase with decreasing hazard penalties becomes more noticeable as SFR increases. For example, availability increases by 794,473% for FT-RMT-QED as t_h decreases from $10^{15} \cdot t_s$ to $10^{6} \cdot t_s$ when SFR is 2.088167×10^{2}/h and t = 100 hours. Similarly, availability increases by 0.36% and 0.44% for FT-RMT-TMR and FT-RMT-TMR-QED, respectively, as t_h decreases from $10^{15} \cdot t_s$ to $10^{6} \cdot t_s$ when SFR is 2.088167×10^{2}/h and t = 100 hours. We note that hazard penalty decrease can be interpreted as an improvement in the acceptance tests' soft error detection capability, and consequently this hazard penalty decrease results in an increase in the availability furnished by a given FT design. Furthermore, assignment of higher hazard penalty for high ASIL levels in D2CyberSoft signifies the criticality of the risk associated with the implemented function, which requires sophisticated FT designs that do not depend solely on acceptance tests for error recover.

Steady-State Availability Evaluations: We also evaluate steady-state availability (i.e., availability as $t \to \infty$) imparted by our implemented SBW designs. Table III presents steady-state availability evaluations for different SFRs when $t_h = 10^{6} \cdot t_s$. These steady-state availability evaluations verify the trends observed in point availability assessments discussed in previous subsections. For example, FT-RMT, FT-RMT-QED, FT-RMT-TMR, and FT-RMT-TMR-QED enable 59% and 1,198% higher availability on average than an NFT design when ECU's SFRs are 2.088167/h and 2.088167×10^{1}/h, respectively, and $t_h = 10^{6} \cdot t_s$. The steady-state availability evaluations verify that sophisticated FT techniques (e.g., FT-RMT-QED) render high availability than the designs with fewer redundancy. For instance, FT-RMT-QED and FT-RMT-TMR impart 19.5% and 272% higher availability, respectively, than FT-RMT when ECU's SFR is 2.088167×10^{3}/h. Results further verify that designs leveraging TMR become paramount as SFR increases. For example, FT-RMT-TMR imparts 211% higher availability than FT-RMT-QED when

ECU's SFR is 2.088167×10^3/h. FT-RMT-TMR provides 0.48% higher availability than FT-RMT-TMR-QED because of additional overhead associated with QED in FT-RMT-TMR-QED, however, FT-RMT-TMR-QED provides 210% higher availability than FT-RMT-QED when ECU's SFR is 2.088167×10^3/h.

We observe that the designs that leverage more sophisticated FT techniques such as FT-RMT-QED and FT-RMT-TMR-QED do not necessarily increase the system availability. Results indicate that FT-RMT-TMR-QED can actually decrease the system availability because of the overhead associated with the technique. We clarify that although FT-RMT-TMR-QED can decrease system availability in certain instances, however, the technique is able to quickly detect and correct computational errors that help in meeting hard real-time deadlines. Hence, both real-time deadlines and desired availability needs to be considered in the design of safety-critical automotive systems.

VII. Conclusions

In this paper, we propose D2CyberSoft—a design automation tool for cybercars that analyzes soft errors and facilitate cybercar designers in selecting designs to alleviate soft errors' impact on availability with minimum cost overhead. We formulate Markov models that provide basis for D2CyberSoft for various fault-tolerant (FT) designs including FT by redundant multi-threading (FT-RMT), FT-RMT enhanced with quick error detection (FT-RMT-QED), FT-RMT with triple modular redundancy (FT-RMT-TMR), and FT-RMT-TMR enhanced with quick error detection (FT-RMT-TMR-QED). We analyze availability of these FT designs for different soft error rates and hazard penalties using D2CyberSoft.

Results reveal that SBW designs leveraging either of FT-RMT, FT-RMT-QED, FT-RMT-TMR, and FT-RMT-TMR-QED provide 39x higher availability on average than a non-fault-tolerant (NFT) design when elapsed time t = 8,760 hours and soft failure rate (SFR) is 2.088167×10^{-4}/h. Furthermore, FT-RMT-QED enables 0.04% higher availability than FT-RMT, which depicts the advantage of quick error detection (QED) for dual modular redundant (DMR) designs, when ECUs' SFR is 2.088167×10^{-1}/h. Results divulge that designs leveraging TMR become more promising at high SFRs as compared to the designs with fewer redundancy. Interestingly, FT-RMT-TMR-QED can actually decrease the system availability as compared to FT-RMT-TMR because of the additional overhead associated with QED, however, FT-RMT-TMR-QED still provides higher availability than FT-RMT-QED. Results reveal that early error detection techniques such as QED, which can be beneficial in meeting real-time deadlines, may become detrimental for availability, especially for designs with redundancy higher than DMR. Results obtained from D2CyberSoft suggest that an optimal cybercar design should incorporate redundancy considering acceptable availability, desired lifetime, cost, expected soft error rates, real-time deadlines, and the safety criticality of the function implemented by the design.

Acknowledgments

This work was supported by the Office of Naval Research (ONR R17460) and SRC GRC Freescale grant (R65000). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ONR and the SRC GRC.

References

[1] D. Skarin and J. Karlsson, "Software Implemented Detection and Recovery of Soft Errors in a Brake-by-Wire System," in *Proc. of EDCC*, May 2008.

[2] ReliaSoft, "ReliaSoft—Empowering the Reliability Professional," 2013. [Online]. Available: http://www.reliasoft.com/

[3] SHARPE, "The SHARPE Tool & the Interface (GUI)," 2013. [Online]. Available: http://people.ee.duke.edu/∼chirel/IRISA/sharpeGui.html

[4] OSATE, "OSATE—Open Source Tools," in *Software Engineering Institute, Carnegie Mellon University*, 2013. [Online]. Available: http://www.sei.cmu.edu/dependability/tools/osate/index.cfm

[5] A. M. Saleh, J. J. Serrano, and J. H. Patel, "Reliability of Scrubbing Recovery-Techniques for Memory Systems," *IEEE Trans. on Reliability*, vol. 39, no. 1, pp. 114–122, April 1990.

[6] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *Proc. of IEEE/ACM MICRO-36*, December 2003.

[7] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions," in *Proc. of IEEE DSN*, June 2007.

[8] J. A. Blome, S. Gupta, S. Feng, S. Mahlke, and D. Bradley, "Cost-Efficient Soft Error Protection for Embedded Microprocessors," in *Proc. of ACM CASES*, October 2006.

[9] E. J. Ossi, "Soft-error mitigation at the architecture-level using berger codes for error detection," Master's thesis, EE Dept., Vanderbilt University, Nashville, Tennessee, 2011.

[10] F. Wang, "Soft error rate determination for nanometer cmos vlsi circuits," Master's thesis, ECE Dept., Auburn University, Alabama, 2008.

[11] R. R. Rao, K. Chopra, D. Blaauw, and D. Sylvester, "An Efficient Static Algorithm for Computing the Soft Error Rates of Combinational Circuits," in *Proc. of IEEE DATE*, March 2006.

[12] R. Rajaraman, J. S. Kim, N. Vijaykrishnan, Y. Xie, and M. J. Irwin, "SEAT-LA: A Soft Error Analysis Tool for Combinational Logic," in *Proc. of IEEE International Conference on VLSI Design*, January 2006.

[13] A. Munir and F. Koushanfar, "D2Cyber: A Design Automation Tool for Dependable Cybercars," in *Proc. of IEEE/ACM Design, Automation & Test in Europe (DATE)*, Dresden, Germany, March 2014.

[14] A. M. Johnson and M. Malek, "Survey of Software Tools for Evaluating Reliability, Availability, and Serviceability," *ACM Computing Surveys*, vol. 20, no. 4, pp. 227–269, December 1988.

[15] H. Lambert, "Use of Fault Tree Analysis for Automotive Reliability and Safety Analysis," in *SAE 2004*, March 2004.

[16] T. Hong, Y. Li, S.-B. Park, D. Mui, D. Lin, Z. A. Kaleq, N. Hakim, H. Naeimi, D. S. Gardner, and S. Mitra, "QED: Quick Error Detection Tests for Effective Post-Silicon Validation," in *Proc. of IEEE ITC*, November 2010.

[17] C. Wilwert, N. Navet, Y.-Q. Song, and F. Simonot-Lion, *Design of Automotive X-by-Wire Systems*. The Industrial Communication Technology Handbook CRC Press, 2005.

[18] Freescale, "MPC5746M: Qorivva 32-bit Multicore MCU for Powertrain Applications," 2013. [Online]. Available: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MPC5746M

[19] Freescale, "e200z4 Power Architecture Core Reference Manual," 2009. [Online]. Available: http://www.freescale.com/files/32bit/doc/ref_manual/e200z4RM.pdf

[20] C. Slayman, "Soft Error Trends and Mitigation Techniques in Memory Devices," in *Proc. of IEEE RAMS*, January 2011.

[21] W. Zhang, X. Fu, T. Li, and J. Fortes, "An Analysis of Microarchitecture Vulnerability to Soft Errors on Simultaneous Multithreaded Architectures," in *Proc. of IEEE ISPASS*, April 2007.