Hardware-Software Interaction in Dependability Modelling of Fault Tolerant System

Yudi Purwantoro[1] Stuart Bennett[2]

Abstract:
The reliability data obtained from the field shows significant deviation from the expectation, especially from those obtained from reliability modelling. This hints that a more acurate model should be employed in the evaluation. A detailed model incorporating a transient fault in the form of hardware-software interaction offers a solution to this problem.

1. Introduction

In reliability evaluation, accuracy of the model relative to the real system is important for the validity of the results. For highly reliable systems, small variation in the model may affect the results significantly. For systems used in safety applications, the regulations specify extremely stringent requirements. For example, FAR/JAR 25.1309 [FAA1, JAA] requires that the system in a civil aircraft be designed such that the probability of failure which prevents the continued safe flight and landing must be extremely improbable. The Advisory Circular 25.1309-1A [FAA2] stressess that to comply with this requirement the system be designed such that the probability of failure must not exceed $1x10^{-9}$ per flight hour. For high availability application, such as safety system in process industry, a requirement for probability of failure of $1x10^{-7}$ or less is not uncommon. In telephony communication, operators demand that the system be able to provide service with interruption of less than three minutes per year [Lap84]. This is equivalent to an unavailability of about $1x10^{-5}$. The field statistical data for the period of 1959-1990 for jet aeroplane show a catastrophic event occurrence rate of about $1x10^{-6}$ per flight hour [Vas96]. Shooman [Sho96] has calculated the failure occurrence rate for avionics software based on the field data and obtained a value of about $1x10^{-7}$ per flight hour. On the other hand, developers convincingly provide reliability data that exceeds the requirement. Such significant difference between the real life data, the requirement and the number provided by the developer has hinted that some aspects of the system which may affect the reliability have been overlooked in the evaluation. Hence it is believed that a more accurate evaluation is needed in order to provide the required level of reliability.

High level of accuracy requires designers to employ a modelling technique which accounts for as many aspects of a system as possible. This leads to the use of a detailed model of system. In line with this requirement, the approach which is widely practised by designers, based on combinatorial methods, in particular the popular reliability block diagram (RBD) technique, needs to be re-examined. RBD technique, although practical because of their simplicity and ease, cannot cater for more than one type of parameters in one model. Interactions between components, particularly between hardware and software in the presence of transient faults such as bit-flip, are not considered. Each component is assumed to fail independently, and usually due to hard faults (permanent faults). Other parameters not included in the model are assumed to be

[1] Department of Automatic Control and Systems Engineering, University of Sheffield, Email: cop97yp@sheffield.ac.uk
[2] Department of Automatic Control and Systems Engineering, University of Sheffield

perfect. Hence, the RBD approach tends to yield optimistic results. This may partially explain the fact that field data revealed higher failure rate than expected.

Transient faults, bit-flip being one of them, are believed to be a significant cause of failure of computer systems. Studies on IBM and VAX computers show that more than 20 percents of software failures are hardware-related [Iye95]. Therefore, in view of the stringent requirement, dependability modelling of safety system should account for these types of faults together with hard faults. Transient faults will manifest in the model in the form of hardware-software interaction. Consequently the designer cannot apply RBD technique but should turn to the more complicated state-space methods, such as Petri nets (PN), stochastic reward nets (SRN) or stochastic activity networks (SAN).

The contribution of this paper is reliability evaluation using a detailed model of computer system. In this approach the model incorporates hard faults in the hardware, transient faults in the hardware, hard faults in the software and the effects of hardware transient faults on the software behaviour. Bit-flip is used as a model of transient fault. The results of detailed model are compared to those of RBD model. Stochastic activity networks is employed to model computer systems which experience permanent faults and bit-flip. Bit-flip is modelled as hardware-software interaction.

2. Bit-flips

Bit-flip is a form of transient faults where one or more bits of a digital device is flipped inversely, i.e. a “1” is flipped to “0” or vice versa. Since bit-flip affects the bit content of device, it may change the code or data contained in that device. This means that a bit-flip can eventually affect the software. Different processors behave differently in the presence of bit-flip because the way instruction code and data are executed or handled is different [Sos96]. In general, as reported by Furtado and Madeira [Fur96] 60% of transient faults in RISC processors cause data errors and 33% cause code (control flow) errors. For CISC processors 18% of transient faults cause data error and 77% cause code error. There is a trade-off between RISC and CISC processors with regard to data to code size ratio. RISC codes tend to be larger than CISC codes. Researchers at NASA report that bit-flips occur more in memories than in the processors [NASA]. In general RISC processors require larger memories than CISC processors do. This means that systems with different processors may have different susceptibility to bit-flips.

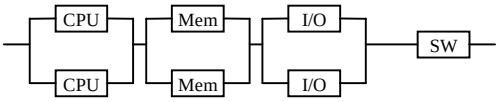
Bit-flips are caused mostly by external disturbances such as heavy ion particles intruding into device, lightning, strong electromagnetic pulses and power supply spikes, and also packaging quality. Depending on the types of system applications, some of these are considered catastrophic when they occur. For example, lightning in particular is viewed as catastrophic disturbance for commercial aircraft. The rate of bit-flip in a device is influenced by the operating environment, the technology of the device (HCMOS, ECL, etc.), circuit density, device area and manufacturing quality. Bit-flips may affect the code section or data section of the software. The probability that it affects the code or the data is influenced by the size of code and data in the application software that resides in the device. Therefore the probability is different between memory and microprocessor or other devices.

Following a bit-flip event, the consequences for the software can take several possibilities. Bit-flip in a code section may result in a software stop, or wrong program flow (or erroneous state). The probability is affected by bit-flip in control section of processor, registers, memory management unit (MMU) or floating point unit (FPU). It is related to the area used by these sections [Kar94]. A bit-flip in the processor has a more severe effect because it is more likely to change the code. A bit-flip in data section may result in fatal output, erroneous output (or data path) or masked by fault tolerant

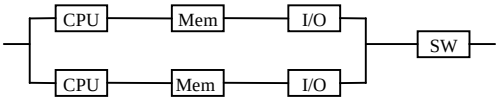
mechanism. The size of memory used by code or data influences the probability of bit-flip in code or data respectively. Bit-flip in I/O is more likely to cause data errors unless it is used to transfer command or instructions to a remote processor.

3. System descriptions

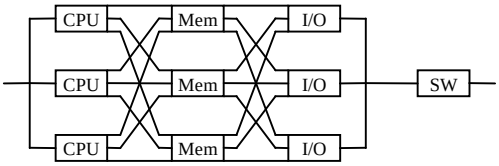
Three popular system configurations represented in block diagrams as shown in Figure 1 are studied. Configuration 1 describes a hot-standby system where each hardware component is dual redundant but they form a single-channel system. The same copy of software runs in redundant hardware devices. If one component experiences hard fault the redundant component takes over with probability one. Failures of software in one component implies the failure of software in the redundant component because the software is the same copy. Although running in redundant devices, the software receives the same data and therefore will fail at the same time. Failure of software or both redundancies brings the system down. When a device experiences bit-flip, the redundant part does not necessarily experience bit-flip. Therefore only software in one device is affected by bit-flip while software in the other devices is not influenced.



(a) Configuration 1: redundant single channel.



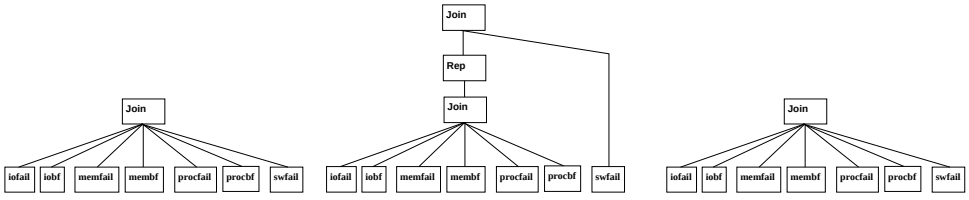
(b) Configuration 2: dual channel.



(c) Configuration 3: triple modular redundant, majority voting

Figure 1 Reliability block diagram models of system configurations

Configuration 2 describes a dual-channel hot-standby system. Each channel is formed by series configuration of CPU, memory and I/O. If one channel fails, the other takes over with probability one to indicate a perfect switching. The software is single version. The failure of the software in one channel implies the failure of the software in the other channel. If one channel experiences a bit-flip the other channel does not necessarily experience a bit-flip.



(a) Configuration 1 (b) Configuration 2 (c) Configuration 3

Figure 2 Stochastic activity networks composed models in detailed modelling

Configuration 3 describes a triple modular redundant system with majority voting. The voter is assumed to be perfect. Each component is triple redundant. Basically this configuration work in similar way to Configuration 1 except that two components of each type are required to function instead of one. As long as there are at least two channels working the system is operational. If one channel fails, the other two vote to give the output. The software is also single version. The failure of the software in one channel implies the failure of the software in the other channels. If one channel experiences a bit-flip the other channels do not necessarily experience a bit-flip.

4. Modelling bit-flip with stochastic activity networks

Bit-flip in memory is used as example to describe how it is modelled. The SAN model of failure for each device is called a subnet and becomes a node in a composed model of the system. A bit-flip event is represented by timed activity MemBf as in Figure 3. A timed activity indicates that bit-flip occurs according to a time distribution function. Because bit-flip is caused by external disturbances whose rate of occurrence cannot be controlled nor deterministically determined, it is assumed that it occurs randomly with constant rate specified by variable *MemBfRate*. It takes the form of exponential distribution. The consequences of a bit-flip take one of six possibilities, all of which define the states of the software. This is indicated by six small circles representing six cases on the right side of activity MemBf. Each of the cases has its own probability function which denotes the likelihood that the case occurs. The first case represents an event when bit-flip occurs in memory but it does not affect the software. This can happen, for example, when the bit of memory is not occupied by the software. The probability that this case is taken is specified by variable *MemBfCov*. The two cases 2 and 3 correspond to the consequences of bit-flip when it affects the code part of the software. The probability that the code is affected by bit-flip is specified via variable *MemCodeBfProb*, and the probability data is affected is (1–*MemCodeBfProb*). Case 2 is the probability that bit-flip affecting the code results in the crash of the software and is specified by *SwStopProb*. Case 3 is the probability that bit-flip on the code causes erroneous state which may be corrected by the available handling mechanism in the software, and is specified by (1–*SwStopProb*). So case 2 returns a probability value of (1–*MemBfCov*)(*MemCodeBfProb*)(*SwStopProb*) and case 3 returns a value of (1–*MemBfCov*)(*MemCodeBfProb*)(1–*SwStopProb*). The other three cases model the consequences of bit-flip when it affects the data part of the software. Case 4 specifies the probability that bit-flip affecting the data will yield an output value that causes fatal actuator movement, and this is specified by variable *FatalProb*. Case 5 models the probability that the output value is not fatal but contains an error that causes a drift in the subsequent calculations. Variable *DriftProb* specifies its probability. Case 6 is the probility that bit-flip in the data has no effect because the error is masked. Overall, case 4 returns a probability value of (1–*MemBfCov*)(1–*MemCodeBfProb*)(*FatalProb*); case 5 returns a probability value of (1–*MemBfCov*)(1–*MemCodeBfProb*)(1–*FatalProb*)(*DriftProb*); and case 6 returns a probability value of (1–*MemBfCov*)(1–*MemCodeBfProb*)(1–*FatalProb*)(1–*DriftProb*).

The execution of activity MemBf indicates that a bit-flip occurs. The activity MemBf is enabled by input gate IG1 when all of the following conditions are true:

i) Not all memories have hard-failed as indicated by the marking on place mem2.
ii) No software has hard-failed, as indicated by the absence of marking on place swfail.
iii) Software has not stopped completely due to either bit-flips or hard faults. This is indicated by the presence of marking on place SwRunBf.
iv) Bit-flip has not exhausted the software. This is indicated by the presence of marking on place SwRunBf.

These conditions generally apply for all three configurations. The difference between each configuration is on the marking of some places. In the above conditions, when the phrase "the presence of marking" is stated, it means that for Configuration 1 and 2 the marking should be at least one, while for TMR configuration it should be at least two.

Output gates og1 to og5 control the marking of places SwStop, StateErr, Fatal, Drift and Masked when the activity MemBf completes. The controlling functions of these gates differ for each configuration because the failure mechanisms differ. The functions of output gates for Configuration 1 and 2 increment the marking of these places when the marking of mem2 reaches zero, while for TMR configuration when the marking of mem2 reaches one. The numbers in the brackets after places SwRunBf and mem2 indicate the number of redundancies in the system and are dependent on the structure of the system.

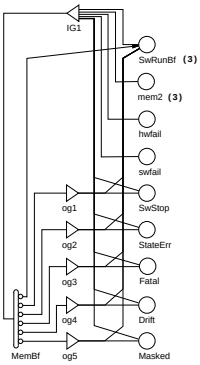
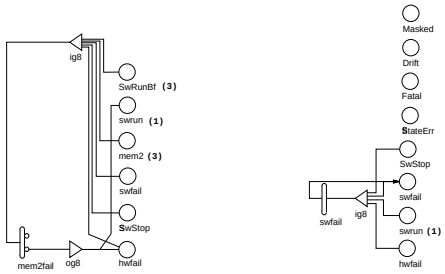


Figure 3 Bit-flip in memory



(a) (b)

Figure 4 Failure due to hard faults in: (a) memory (b) software.

5. Modelling hard faults with stochastic activity networks

Hard faults are associated with wear out or systematic failure. The rate of failure is assumed constant. Failure rate is usually specified otherwise as mean time to failures (MTTF) which is the inverse of failure rate.

Figure 4 shows the SAN models for memory and software failures. The failure of memory is modelled by timed activity mem2fail. The failure time of memory follows an exponential distribution with a constant rate of *MemFRate*. In this model there are two possibilities when a memory fails. If the system is a hot-standby with perfect switching, when the first memory fails, case 1 is selected and a probability value of 1 is returned, otherwise a value of 0 is returned. If the switching is not perfect, i.e. if the hot-spare takes over the failed part with a probability of success *memcov*, then case 1 returns a value *memcov* and case 2 returns a value (1–*memcov*). The failure rate of a hot-standby system or a triple modular redundancy system is multiplied by the number of redundancies which have not yet failed (i.e. multiplied by the marking of place mem2).

The activity mem2fail is enabled by input gate ig8 when all of the following conditions are true.

(i) No hardware has hard-failed. This is indicated by the absence of marking on place hwfail and the presence of marking on place mem2.

(ii) No software has hard-failed. This is indicated by the absence of marking on place swfail and the presence of marking on place swrun.

(iii) No bit-flip has stopped the software in all devices. This is indicated by the absence of marking on place SwStop and the presence of marking on place SwRunBf.

Hard faults on hardware always cause software stop. This is modelled via output gate *og8*

by clearing the marking of place swrun. Software failure is modelled by timed activity swfail. The failure time follows an exponential distribution with a rate of *SwFRate*. The activity swfail is enabled by input gate ig8 when all the following conditions are true.

(i) No hardware has hard-failed. This is indicated by the absence of marking on place hwfail.
(ii) No software has hard-failed. This is indicated by the absence of marking on place swfail and the presence of marking on place swrun.
(iii) No bit-flip has occured to all devices which stops the software. This is indicated by the absence of marking on place SwStop.

6. Analysis using reliability block diagram (RBD)

Reliability analysis using this method remains the simplest and most efficient way. It is intuitive in that designers can quickly figure out which components contribute most significantly to system reliability. A few assumptions are made when using RBD:

i) Component fails independently, i.e. failure of one component does not affect the operation of others.
ii) When a fault occurs in a component, the component fails absolutely, i.e. the states of component are either good or failed
iii) Cause of failure is limited to one fault; failures are of hard fault type; no transient faults are considered; hence only one type of parameter can be studied.
iv) Redundant part takes over the failed part with a perfect switching.

The equations in Table 1 are derived from the RBD models in Figure 1. $R(t)$ denotes the reliability (probability of successful operation) of system while $F(t)$ denotes the unreliability (probability of failed operation). Unreliability is evaluated as function on time, I/O failure rate (*IoFRate*), memory failure rate (*MemFRate*), processor failure rate (*ProcFRate*) and software failure rate (*SwFRate*).

Table 1 Reliability equations for RBD models

Configuration 1 $F_{\text{Config1rbd}}(t) = 1 - R_1(t)R_2(t)R_3(t)R_{\text{SW}}(t)$ where $R_1(t) = 1-[1-e^{-\lambda_{\text{CPU}}t}]^2$ $R_2(t) = 1-[1-e^{-\lambda_{\text{Mem}}t}]^2$ $R_3(t) = 1-[1-e^{-\lambda_{\text{IO}}t}]^2$ $R_{\text{SW}}(t) = e^{-\lambda_{\text{SW}}t}$ **Configuration 2** $F_{\text{Config2rbd}}(t) = 1 - R_{\text{HW}}(t)R_{\text{SW}}(t)$ where $R_{\text{HW}}(t) = 1-[1-e^{-\lambda_{\text{CPU}}t}e^{-\lambda_{\text{Mem}}t}e^{-\lambda_{\text{IO}}t}]^2$ $R_{\text{SW}}(t) = e^{-\lambda_{\text{SW}}t}$	**Configuration 3** $F_{\text{TMRrbd}}(t) = 1 - R_1(t)R_2(t)R_3(t)R_{\text{SW}}(t)$ where $R_1(t) = 3[e^{-\lambda_{\text{CPU}}t}]^2 - 2[e^{-\lambda_{\text{CPU}}t}]^3$ $R_2(t) = 3[e^{-\lambda_{\text{Mem}}t}]^2 - 2[e^{-\lambda_{\text{Mem}}t}]^3$ $R_3(t) = 3[e^{-\lambda_{\text{IO}}t}]^2 - 2[e^{-\lambda_{\text{IO}}t}]^3$ $R_{\text{SW}}(t) = e^{-\lambda_{\text{SW}}t}$

7. Reliability Analysis

7.1. Aims

The analysis will be done in two ways. First, in section 7.3.1, the RBD results for the three configurations are compared and analysed as function of time, I/O failure rate, memory failure rate, processor failure rate and software failure rate. It suggests which parameters and at what values are critical to system reliability, in addition to showing which configuration is most reliable. The results become the upper bound for reliability, i.e. results from using other methods will not exceed the results from using reliability block diagram.

Second, comparison between RBD and stochastic activity networks models is presented in section 7.3.2. It aims at studying the significance of bit flip contribution to system unreliability. The results reveals the worthiness of conducting a detailed evaluation in respect of the requirements. In the detailed model, parameters other than "traditional" failure rate, which manifest in the form of hardware-software interaction, are considered. The study suggests how much a certain value of bit flip parameters contributes to the unreliability. This is done by evaluating unreliability as function of *IoCodeBfProb*, *MemCodeBfProb* and *ProcCodeBfProb*.

7.2. Parameter values

Parametric studies are done by varying one parameter over a reasonable range while holding other parameters constant at values obtained from field experience as found in literatures or popular product data sheet. For example, when studying the effect of memory failure rate on reliability, *MemFRate* spawns between 10^{-8} to 10^{-4} per hour while *SwFRate* is fixed at $1x10^{-6}$ per hour, *ProcFRate* at $1x10^{-6}$ per hour, *ProcBfRate* at $2.5x10^{-5}$ per hour, etc. The following parameters are common for all configurations and all studies:

- Hard failure rates:
  - Varying: $10^{-8} – 10^{-4}$ per hour
  - Fixed:
    MemFRate = $1x10^{-6}$ per hour
    IoFRate = $2x10^{-6}$ per hour
    ProcFRate = $1x10^{-6}$ per hour
    SwFRate = $1x10^{-6}$ per hour
- Bit-flip rates:
  IoBfRate = $1.25x10^{-4}$ per hour
  MemBfRate = $1.25x10^{-4}$ per hour
  ProcBfRate = $2.5x10^{-5}$ per hour
- Probability that bit-flip affects the code:
  IoCodeBfProb = 0.25
  MemCodeBfProb = 0.75
  ProcCodeBfProb = 0.9
- Probability associated with bit-flip consequences:
  - in code:
    SwStopProb = 0.8
  - in data:
    DriftProb = 0.5
    FatalProb = 0.2
- Probability that bit-flip is covered:
  IoBfCov = 0.99

MemBfCov = 0.99
ProcBfCov = 0.99

7.3 Analysis of results

7.3.1. Comparing configurations

Figure 5 compares the unreliability of Configuration 1, Configuration 2 and Configuration 3 for operating time between 1 hour and 15000 hours. The results show that Configuration 1 is consistently the most reliable configuration for the whole duration. It is interesting to note that Configuration 2 is more reliable than Configuration 3 by a very small margin. These results suggest that the higher number of components of Configuration 3 leads to reduced reliability. Although Configuration 3 can tolerate more component failures, the more count of components means higher probability of failures. Hence the number of component count should be given a real consideration by designer in s/he wants to obtain a high reliability system. This is true eventhough redundancies are present in the system if the redundancies incorporate imperfect switching or coverage. For the three configurations in general, the reliability is limited by the reliability of software due to its single version.

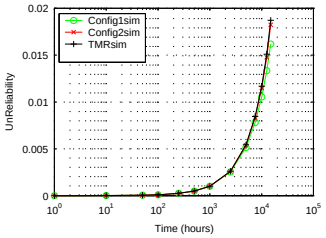


Figure 5 Unreliability vs. time obtained from reliability block diagram.

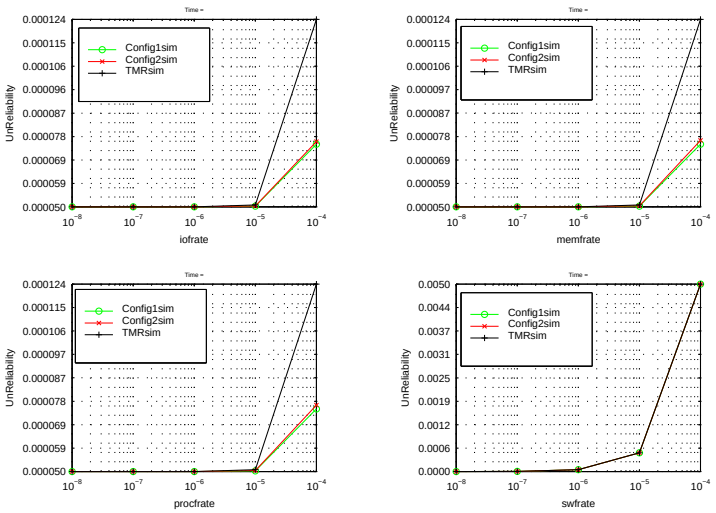


Figure 6 Unreliability vs. failure rate obtained from reliability block diagram.

In Figure 6 comparison is made when unreliability is evaluated as function of *IoFRate* at *t*=50 hours. Configuration 1 is again most reliable. While Configuration 1 and Configuration 2 generally exhibit similar characteristics, Configuration 3 presents a notable deviation for *IoFRate*=10^{-4}, for which the unreliability jumps by almost 150% compared to unreliability at *IoFRate*=10^{-5}. The unreliability of Configuration 1 and

Configuration 2 increases by almost 50% at the same value of *IoFRate*.

Figure 6 also presents the comparison when unreliability is evaluated as function of *MemFRate* at *t*=50 hours. The results generally show similar characteristics to those as function of *IoFRate*. Unreliability at t=50 hours as function of *ProcFRate* shows characteristics as for *MemFRate*. The results of unreliability evaluated at t=50 hours as function *SwFRate* suggest that due to single version of the software, unreliability of the three configurations are highly sensitive to the change in *SwFRate*. Although Configuration 1 is still the most reliable configuration, the difference among the three is not as noticable as when other parameters change.

The results of five studies as shown above confirm the superiority of Configuration 1 with respect to hard failures while Configuration 2 is slightly more reliable than TMR configuration. It has also been confirmed that a single component affects the system reliability more significantly. The study also confirms that evaluating reliability using models with hard faults only does not suggest a lot of ways to improve the design.

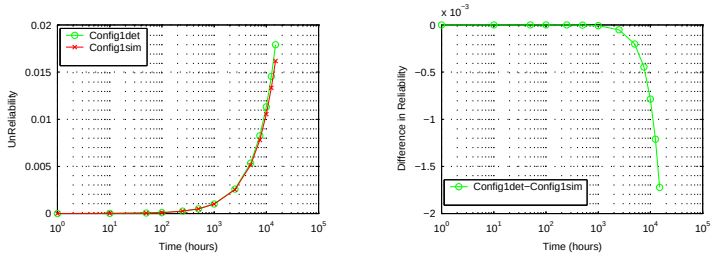


Figure 7 Unreliability vs. *time* comparing RBD and SAN model

7.3.2. RBD vs. SAN model

7.3.2.1. Configuration 1

Figure 7 compares the results obtained from using RBD model detailed SAN models for operating time between 1 hour and 15000 hours. It can be seen that bit-flips do not affect the reliability for time before 50 hours, but after this time bit-flips impose strong influence on the reliability. As time passes the effect of bit-flip cannot be ignored.

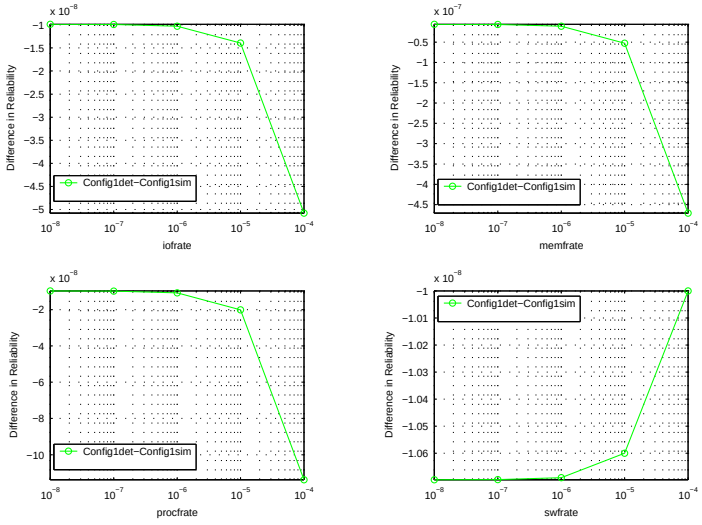


Figure 8 Difference in reliability vs. failure rate comparing RBD and SAN model

Figure 8 compares the difference in reliability (ΔR) as function of *IoFRate* at *t*=50 hours. The presence of bit-flip in general reduces the reliability equally at low and high failure rate of I/O except when the failure rate goes above 10^{-5}. The figure also shows ΔR as function of *MemFRate*. The presence of bit-flip reduces the reliability less significantly at low than at high failure rate of memories. ΔR as function of *ProcFRate* is shown. The presence of bit-flip in general reduces the reliability equally at low and high failure rate of CPU except when the failure rate goes above 10^{-6}. The last plot compares ΔR as function of *SwFRate*. The presence of bit-flip in general reduces the reliability equally at low and high failure rate of software.

7.3.2.2. Configuration 2

Figure 9 compares the results obtained from using detailed and simple SAN models for operating time between 1 hour and 15000 hours. The results readily show that bit-flips do not affect the reliability for time before 50 hours, but after this time bit-flips impose strong influence on the reliability. As time passes the effect of bit-flip cannot be ignored although it is less than that for Configuration 1.

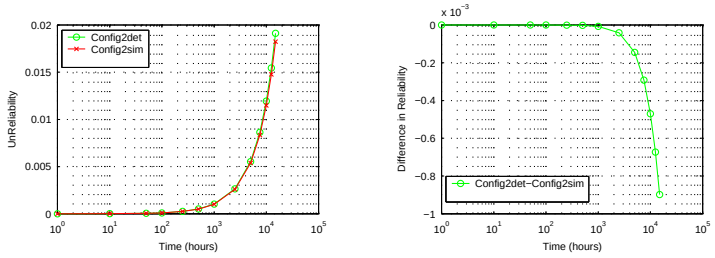


Figure 9 Unreliability vs. *time* comparing RBD and SAN model

Figure 10 compare ΔR as function of *IoFRate* at *t*=50 hours. The reliability is less influenced by bit-flips at low failure rate than at high failure rate. In general the effect of bit-flip is significant for high reliable systems where unreliability should not exceed 10^{-9}. When the failure rates of memory and CPU are varied, the results show similar characteristics to the results of varying the failure rate of I/O. Figure 10 also depicts ΔR as function of *MemFRate*, *ProcFRate* and. *SwFRate*. The presence of bit-flip in general reduces the reliability equally at low and high failure rate of software.

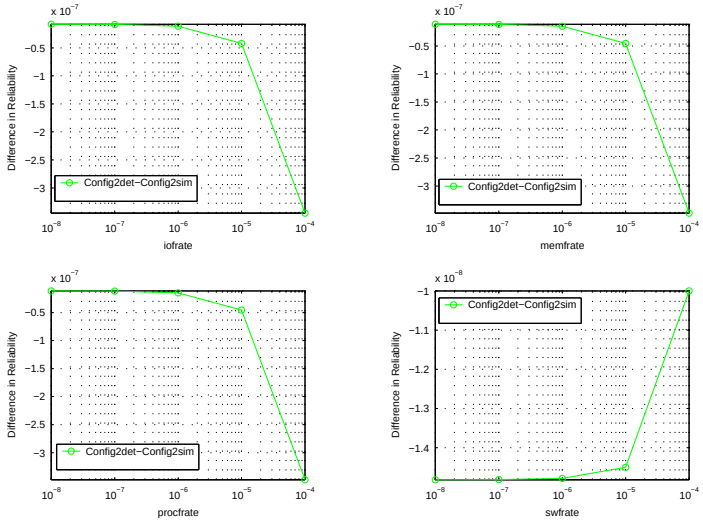


Figure 10 Difference in reliability vs. failure rate comparing RBD and SAN model

7.3.2.3. Configuration 3

Figure 11 compares the results obtained from using detailed and simple SAN models for operating time between 1 hour and 15000 hours. The results readily show that bit-flips do not affect the reliability for time before 50 hours, but after this time bit-flips impose strong influence on the reliability. As time passes the effect of bit-flip cannot be ignored.

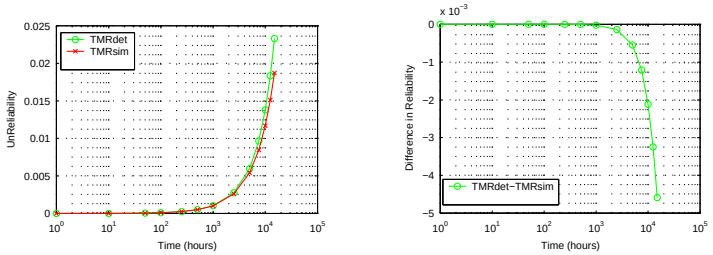


Figure 11 Unreliability vs. *time* comparing RBD and SAN model

Figure 12 compare ΔR as function of *IoFRate* at *t*=50 hours. The reliability is less influenced by bit-flips at low failure rate than at high failure rate. The difference in the reliability between the two models is bigger than that for other configurations. ΔR as function of *MemFRate* and *ProcFRate* indicates that the presence of bit-flip in general reduces the reliability equally at low and high failure rate of memories except when the failure rate goes above 10^{-6}. The presence of bit-flip in general reduces the reliability equally at low and high failure rate of CPU except when the failure rate goes above 10^{-6}. It can be seen from the last plot that the presence of bit-flip in general reduces the reliability equally at low and high failure rate of software. The effect of bit-flip is more notable for this configuration than for the other two.

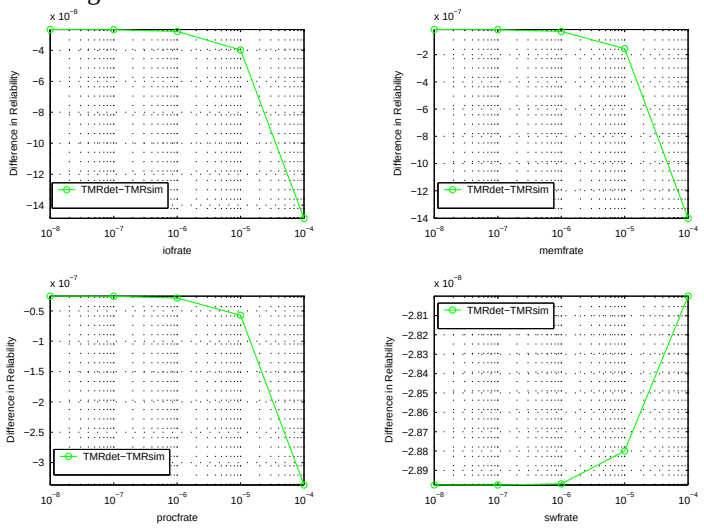


Figure 12 Difference in reliability vs. failure rate comparing RBD and SAN model

8. Conclusion

During preliminary evaluation in the system development or evaluation at system level, reliability block diagram is an important tool. However, as a designer wants to obtain accurate and a “real” reliability number, he has no choice but to use a detailed model, including bit-flip, and consequently to use SAN or other state space methods. The study

has shown that bit-flip as one form of transient faults affects system reliability. If other forms of transient faults are included in the model it is expected that the results may reflect the failure rate obtained from the field. Hence the use of detailed model which accounts for hardware-software interaction should be considered when evaluating the reliability of of a highly reliable system.

The price for using detailed model is long computation time and large storage space. Carefull design of model through good failure specification may lead to significant reduction in both time and space. Even when this is not achievable, designer should always go into a detailed model.

9. References

[CRH95] CRHC, *UltraSAN User's Manual Version 3.0*, University of Illinois, Urbana-Champaign, 1995.

[FAA1] FAA, *Federal Aviation Regulations Part 25*, latest revision.

[FAA2] FAA, *AC25.1309-1A System Design and Analysis*, latest revision.

[Fur96] P. Furtado and H. Madeira, "Fault injection evaluation of assigned signatures in a RISC Processor", in *Dependable Computing - EDCC-2*, Hlawiczka, A., Silva, JG., Simoncini, L. (eds), Springer, Berlin, 1996, pp. 55-72.

[Iye95] R.K. Iyer and I. Lee, "Measurement-based analysis of software reliability", in *Handbook of Software Reliability Engineering* by M.R. Lyu (ed.), McGraw Hill, 1995, pp. 303-358.

[JAA] JAA, *Joint Aviation Regulations Part 25*, latest revision.

[Kar94] J. Karlsson, "Using heavy ion radiation to validate fault handling mechanisms", *IEEE Micro*, 14(1):8-23, February 1994.

[Lap84] JC. Laprie, "Dependability evaluation of software systems in operation", *IEEE Transactions on Software Engineering*, SE-10(6):701-14, Nov. 1984

[NASA] http://flick.gsfc.nasa.gov/radhome

[Rim94] M. Rimen, J. Ohlsson and J. Torin, "On microprocessor error behavior modeling", *Proceedings of the 24rd Annual International Symposium on Fault-Tolerant Computing*, 1994, pp. 76-85.

[Sho96] M.L. Shooman, "Avionics Software Problem Occurrence Rates", *Proceedings FTCS-26*, IEEE Computer Society, 1996, pp. 55-64.

[Sos96] J. Sosnowski and A. Kusmierczky, "Pseudorandom testing of microprocessors at instruction/data flow level", in *Dependable Computing - EDCC-2*, Hlawiczka, A., Silva, JG., Simoncini, L. (eds), Springer, Berlin, 1996, pp. 246-263.

[Str99] R. Stroph, *Acceptance Test Design for Fault Tolerant Digital Control Systems*, PhD thesis, Department of Electronics, University of York, 1999.

[Vas96] A.G. Vassakis, "Safety assessment, reliability, and the probability operation diagram", *IEEE Transactions on Reliability*, 45(1):90-94, March 1996.