Técnicas de Tolerância a Falhas em uma Plataforma para Prototipagem Rápida Usando Microcontroladores

Kleber Kruger[1], **Fábio Iaione**[1]

[1]Faculdade de Computação – Universidade Federal de Mato Grosso do Sul (UFMS)
CEP 79070-900 – Campo Grande – MS – Brazil

Abstract. *This paper describes the implementation of fault tolerance techniques (based on data and processing redundancy) in programming of a rapid prototyping platform using microcontrollers. To evaluate performance of these techniques was used a fault injector software and a weather station system as a case study. Experiments simulated faults in sensor readings and faults in SRAM memory regions of the weather station. Finally, the fault-tolerant system performance is presented in comparison with the non-fault-tolerant system, considering incidence of failures, processing time, memory and power consumption.*

Resumo. *Neste trabalho implementamos técnicas de tolerância a falhas (baseadas na redundância de dados e de processamento) na programação de uma plataforma para prototipagem rápida usando microcontroladores. Para avaliar o desempenho das técnicas foi utilizado um injetor de falhas por software e um sistema de estação meteorológica como estudo de caso. Os testes simularam falhas nas leituras dos sensores e falhas nas regiões de memória SRAM da estação meteorológica. Ao final, são apresentados os resultados de desempenho do sistema sem tolerância a falhas em comparação com o sistema tolerante a falhas, considerando incidência de defeitos, tempo de processamento, consumo de memória e de energia.*

1. Introdução

Com a expansão da computação ubíqua, o uso de sistemas computacionais tem crescido nos últimos anos [Chetan et al. 2005], uma vez que alguns equipamentos antes construídos com pouco ou nenhum recurso computacional tornaram-se mais sofisticados, incorporando algum tipo de sistema embarcado. Os sistemas embarcados, apesar do baixo custo, precisam tolerar falhas, pois erros causados por falhas intrínsecas [Huang et al. 2005], falhas físicas, *bugs* de software ou efeitos do meio ambiente, tais como o fenômeno *bit-flip* [Velazco and Rezgui 2000], podem causar transtornos, danos e prejuízos financeiros [Patterson and Hennessy 2005]. Infelizmente, danos graves causados por falhas em equipamentos já ocorreram no passado [Singh and Murugesan 1990] e esses fatos destacam a importância da tolerância a falhas.

O objetivo deste trabalho foi implementar técnicas de tolerância a falhas na programação de uma plataforma de prototipagem rápida com microcontroladores, utilizando-se uma estação meteorológica como estudo de caso, e avaliar o desempenho dessas técnicas considerando a utilização de memória, o tempo de processamento e o consumo de energia.

2. Tolerância a Falhas

O princípio básico da construção de sistemas tolerantes a falhas[1] é prover recursos extras (redundância) para evitar a ocorrência de defeitos quando uma falha ocorre [Laprie et al. 1990]. Segundo Dubrova [Dubrova 2007] *et al*, uma das formas de se aplicar a redundância é sob os dados e o processamento de um programa.

2.1. Redundância de Dados

Consiste em replicar os dados do programa em diferentes locais ou adicionar dados para permitir a correção ou apenas a detecção de erros. Nos sistemas embarcados por exemplo, os dados de um programa podem ser gravados em diferentes locais, como memória SRAM, arquivo em disco (flash) ou qualquer outro dispositivo de armazenamento [Kruger and Iaione 2013].

2.2. Redundância de Processamento

A redundância de processamento repete a execução do código no tempo. Essa técnica evita custos adicionais de hardware, mas aumenta o tempo necessário para realizar um processamento. Esta técnica permite principalmente a detecção de falhas transientes, porque diferentes resultados são um forte indício de uma falha transitória [Nelson 1990].

3. Metodologia

Utilizou-se como ponto de partida o sistema embarcado (*hardware* e *firmware*) de uma estação meteorológica usada para coleta de parâmetros climáticos em lavouras. O *firmware* funciona lendo periodicamente (a cada 5 min) todos os sensores da estação meteorológica e gravando o conjunto de leituras, mais um código CRC, em um arquivo binário na memória *flash*. Além dos códigos CRC, que permitem a detecção de erros no arquivo gravado, o microcontrolador possui um temporizador *watchdog* que reinicia o dispositivo quando travado deixa de recarregar o temporizador. O *hardware* da estação meteorológica foi baseado na plataforma de prototipagem rápida para microcontroladores mbed [mbed 2014], modelo NXP LPC1768. O módulo NXP LPC1768 é composto por um núcleo ARM Cortex-M3 de 32 bits e 96 MHz de *clock*, memória *flash* com capacidade de 512 kB, e memória SRAM de 64 kB (32 kB destinados ao *firmware*, e dois blocos de 16 kB destinados aos controladores internos do microcontrolador: USB, DMA e ethernet).

Com base no *firmware* original da estação meteorológica, um outro foi desenvolvido empregando tolerância a falhas por meio da redundância de dados e de processamento, as duas técnicas mais viáveis em sistemas embarcados de baixo custo. Para testar o desempenho das técnicas de tolerância utilizadas, foi desenvolvido um sistema para injeção de falhas e monitoramento da estação meteorológica, detalhadamente descrito em [Kruger and Iaione 2013]. Esse sistema é composto por três partes: a biblioteca de injeção de falhas, o *firmware* monitor e o programa monitor de testes. A biblioteca de injeção de falhas, denominada *FaultInjector*, atua alterarando dados aleatoriamente nas regiões de memória SRAM do microcontrolador e causando erros no programa sob teste, simulando falhas provocadas por eventos ambientais transientes, como o *bit-flip*

[1]Nesse trabalho é utilizada a terminologia definida em [Nelson 1990], segundo a qual uma falha (domínio físico) gera um erro (domínio da informação) que provoca um defeito (domínio do usuário).

[Velazco and Rezgui 2000] e a interferência eletromagnética. O *firmware* monitor, executado em outro módulo mbed (mbed monitor), serve para emular os sinais (pulsos) de pluviometria e anemometria, e para reiniciar a estação meteorológica quando defeitos são detectadas pelo programa monitor. O programa monitor, executado em um computador pessoal, monitora os resultados dos testes em tempo de execução, analisando o arquivo de dados gravado pela estação. Ao final da execução de um número predefinido de testes, um relatório detalhado é gravado no computador mostrando a quantidade de falhas ocorridas juntamente com os tipos das falhas ("arquivo de dados não encontrado", "dados incorretos" e "CRC incorreto").

Na Figura 1 mostra-se um fluxograma do programa monitor de testes, usado para analisar os resultados dos ciclos de leitura da estação meteorológica em tempo de execução. Cada ciclo de leitura corresponde aos passos: 1) verificar o horário atual, 2) ler os 9 sensores da estação meteorológica (temperatura do ar, umidade do ar, pluviometria, anemometria, temperatura do solo, umidade do solo, irradiação solar, molhamento foliar e tensão da bateria) e 3) gravar os dados em um arquivo binário na memória flash.

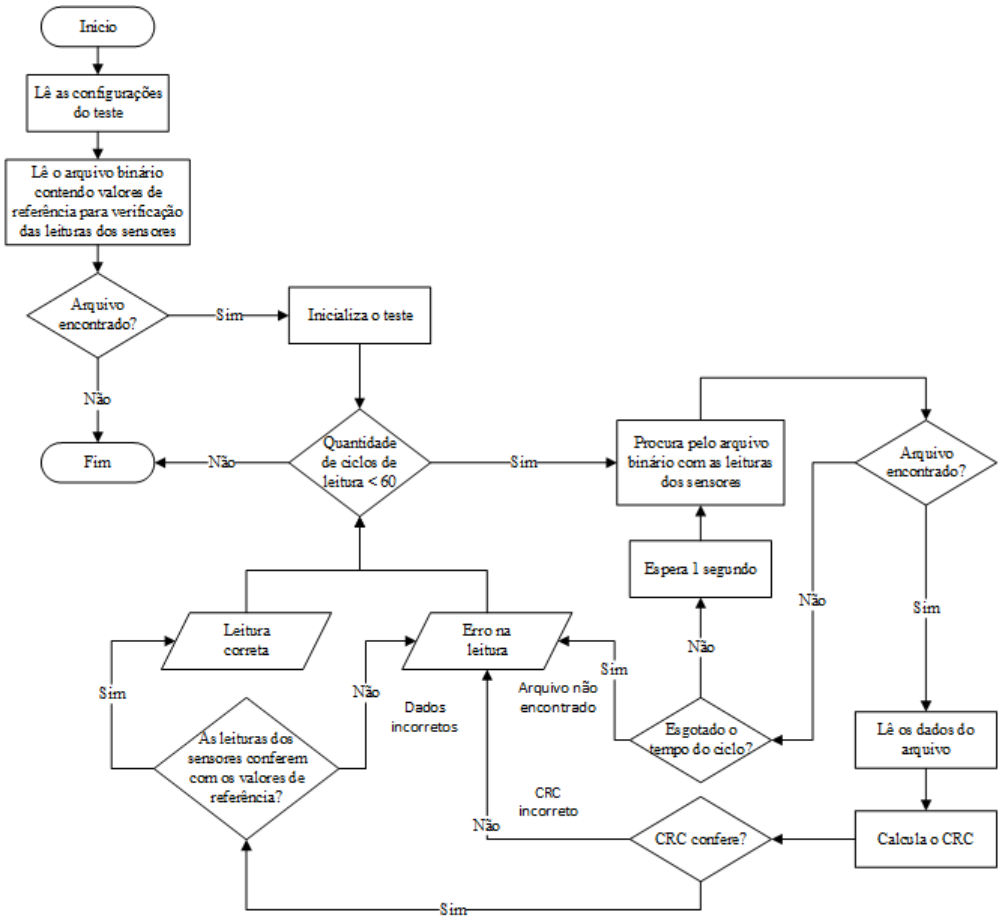


Figura 1. Fluxograma do programa monitor de testes.

A cada ciclo de leitura, o programa monitor procura o arquivo binário contendo os dados das leituras. Este arquivo é escrito periodicamente na memória *flash* pela estação meteorológica. Se em um tempo estipulado (tempo total de um ciclo de leitura somado a um valor de tolerância de 1 segundo) o arquivo não for encontrado, o programa monitor

identifica que houve um defeito durante a execução do teste, provavelmente resultado de uma falha que causou um defeito no sistema. Caso contrário, os dados do arquivo binário escrito pela estação meteorológica são lidos e verificados pelo programa monitor, que em seguida, apaga o arquivo. Para verificar o arquivo, o programa confere primeiramente sua integridade, calculando o código CRC e comparando-o ao código CRC encontrado no arquivo. Após isso, o valor de cada parâmetro climático contido no arquivo é comparado com os valores do arquivo de referência. Esse processo de comparação aceita pequenas diferenças entre os valores de referência e os valores lidos pela estação, conforme mostra a Tabela 1. Isso é necessário devido às diferenças que podem ocorrer naturalmente entre leituras consecutivas de um conversor analógico-digital, com uma tensão de entrada fixa. Para permitir essa comparação, parâmetros climáticos invariáveis foram simulados através de resistências, tensões e frequências invariáveis que substituíram os sensores. Quando um defeito é encontrado, o programa monitor envia uma mensagem pela porta serial (virtual) para o mbed monitor, que reinicia a estação meteorológica. O programa termina quando atinge a quantidade de ciclos de leitura desejada.

Além da injeção de falhas em regiões de memória do microcontrolador, alguns dos testes realizados neste trabalho simularam a presença de falhas nas leituras dos sensores. Essas falhas em uma situação real podem ocorrer principalmente devido à interferência eletromagnética no momento da leitura de um sensor, gerando valores incorretos. Logo, para simular esses efeitos, o método responsável pela leitura dos sensores da estação meteorológica (*readSensors*) foi alterado em ambos os *firmwares*, com e sem tolerância a falhas, de forma que, após a leitura de cada sensor, o método *injectFaultAtFloat* da biblioteca *FaultInjector* seja executado. Este método permite de uma forma configurável fazer a alteração de um dado, substituindo o valor original por um valor aleatório.

Parâmetro	Variação	Emulação
Temperatura do ar	5%	Não emulado (SPI)
Umidade do ar	5%	Não emulado (SPI)
Pluviometria	30%	Sinal de frequência (3,33 Hz)
Anemometria	5%	Sinal de frequência (1,66 Hz)
Temperatura do solo	5%	Tensão fixa
Umidade do solo	20%	Tensão fixa
Irradiação solar	5%	Tensão fixa
Molhamento foliar	5%	Resistência fixa
Tensão da bateria	10%	Não emulado (medida real)

Tabela 1. Variação máxima permitida para cada parâmetro climático.

4. Técnicas de Tolerância a Falhas Utilizadas

Neste trabalho utilizou-se redundância de dados e de processamento. Para implementar a redundância de dados no *firmware*, todas as variáveis foram triplicadas, mantendo as cópias dos dados em diferentes regiões da memória. O valor de cada variável foi obtido mediante um sistema de votação. Considerando-se três cópias dos dados, representadas por x_1, x_2 e x_3, esses valores podem sofrer alterações devido a ocorrência de falhas, mas se uma das cópias é corrompida, as outras duas mantém os valores corretos da informação e o erro é evitado. Nas situações em que duas das três cópias são alteradas, se os valores

dos dados modificados forem diferentes (situação mais provável) o sistema então assume um estado de erro e toma uma medida de correção, refazendo a operação (redundância de processamento) ou apenas evitando a propagação do erro. A redundância de processamento foi implementada pela reexecução das funções até suas tarefas serem realizadas com sucesso ou o limite de tentativas ser atingido.

O *firmware* da estação meteorológica lê os sensores utilizando a redundância de processamento para obter uma média, desprezando leituras com valores acima de uma variação predefinida. Com isso, o valor da leitura de um sensor, em vez de ser obtido por uma única leitura sujeita a erros, é calculado pela média de um conjunto de leituras, efetuadas em intervalos de i milissegundos e submetidas ao método *avg*. O método *avg* calcula a média dos valores de uma amostra A de tamanho n, desprezando os valores que estejam fora de uma variação v, entre quaisquer elementos que pertencem a média. Este método retorna a média calculada ou o valor NaN, quando os dados analisados não permitem um resultado confiável. Um resultado é classificado como confiável quando, dos n elementos da amostra A, s estão dentro da variação v. O valor da variação v, o valor de s, o tempo do intervalo i entre as leituras e o tamanho n das amostras são configuráveis, tendo como restrições $v \geq 0$ e $n/2 < s \leq n$.

5. Testes Realizados

Todos os testes executados neste trabalho continham, cada um, 60 ciclos de leitura. O primeiro conjunto de testes injetou no *firmware* tolerante a falhas somente falhas nas leituras dos sensores. O objetivo foi analisar a capacidade de tolerância a falhas do método *avg*, juntamente com a redundância de processamento sobre ela.

O segundo conjunto de testes utilizou a biblioteca *FaultInjection* para injetar falhas nas regiões de memória SRAM do microcontrolador, com o objetivo de simular eventos ambientais externos e avaliar o desempenho dos *firmwares* sem tolerância a falhas e com tolerância a falhas. A quantidade de falhas injetadas (endereços de memória SRAM alterados) em cada ciclo foi de 0, 1, 2, 4, 8, 16, 32, 64, 128 e 256. Os testes foram realizados em duas configurações de injeção de falhas: injeção apenas na memória SRAM destinada ao *firmware* (32 kB), e injeção em toda memória SRAM (64 kB). O terceiro conjunto de testes, similar ao segundo, diferiu-se pela injeção adicional de 25% de falhas nas leituras dos sensores.

Para permitir uma grande quantidade de testes em um curto período de tempo, o intervalo entre cada ciclo de leitura da estação meteorológica foi reduzido de 5 min para 10 segundos. Este tempo corresponde a duração máxima (com a adição de uma pequena margem de segurança) de um ciclo de leitura completo em situações com elevadas taxas de injeção de falhas.

6. Resultados

Nas seções a seguir são apresentados os resultados obtidos nos testes realizados.

6.1. Injeção de falhas nas leituras dos sensores

Neste conjunto de testes injetou-se falhas em 10%, 20%, 25%, 30%, 40% e 50% das leituras dos sensores e foi realizado sob duas configurações: na primeira, cujos resultados aparecem na Figura 2, o método *avg* foi configurado com n = 6 e um percentual de leituras

aceitáveis de 66,6% (4 leituras dentro da faixa aceitável de um total de 6), e na segunda, $n = 4$ e percentual de leituras aceitáveis de 75% (3 de 4) (Figura 3). A escolha desses valores baseou-se principalmente no tempo adicional gerado pela redundância de processamento. Os valores definidos permitem amostras com dados suficientes para expressar um resultado confiável e, em contrapartida, não aumentam demasiadamente o tempo de processamento.

As colunas do gráfico da Figura 2 e da Figura 3 contém os valores médios obtidos de 30 testes individuais, cada um com 60 ciclos de leitura. Os valores exibidos nas colunas empilhadas correspondem ao número de vezes que o método *avg* tolerou falhas, ao número de vezes que a redundância de processamento tolerou falhas, e a quantidade total de erros ocorridos durante a execução do programa.

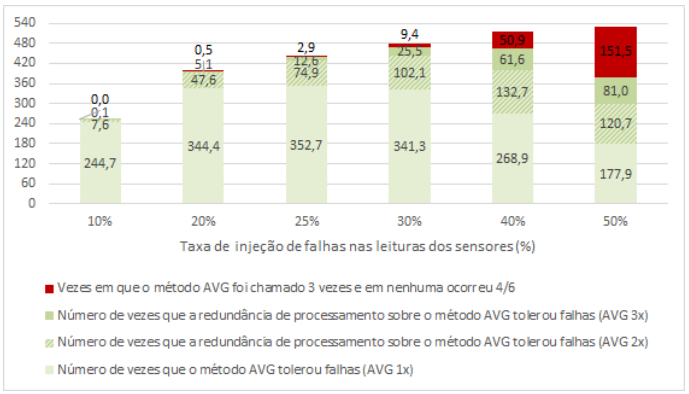


Figura 2. Resultados dos testes de injeção de falhas nas leituras dos sensores, utilizando-se um percentual de leituras aceitáveis de 66,6%.

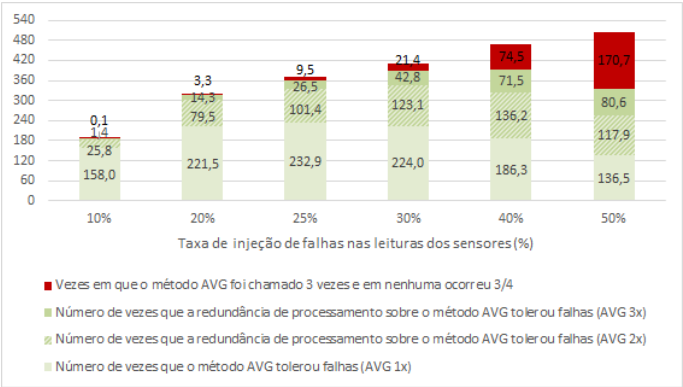


Figura 3. Resultados dos testes de injeção de falhas nas leituras dos sensores, utilizando-se um percentual de leituras aceitáveis de 75%.

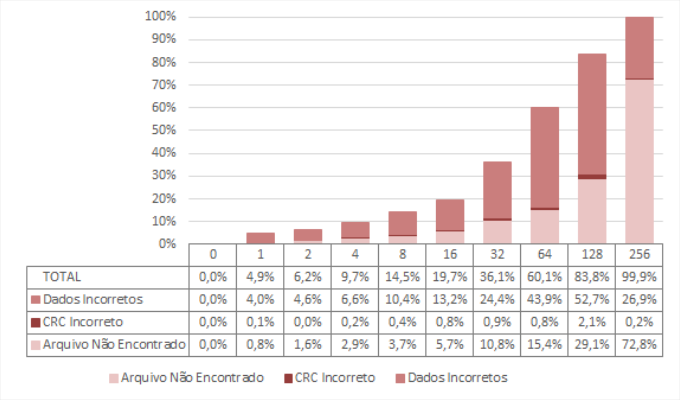
6.2. Injeção de falhas na memória SRAM

Este conjunto de testes utilizou a biblioteca *FaultInjection* [Kruger and Iaione 2013] para alterar dados nas regiões de memória simulando fenômenos ambientais aleatórios sobre o equipamento. Na Seção 6.2.1 mostra-se os resultados quando as falhas foram injetadas somente na região de memória SRAM destinada ao firmware, e na Seção 6.2.2, quando as falhas foram injetadas na SRAM destinada ao firmware e também na SRAM destinada aos controladores internos do microcontrolador. Cada ciclo de leitura apresentou

três diferentes tipos de defeitos: “dados não encontrados”, “CRC incorreto” e “dados incorretos”. Cada valor mostrado nas Figuras 4 a 11 corresponde à média de um conjunto de 30 testes.

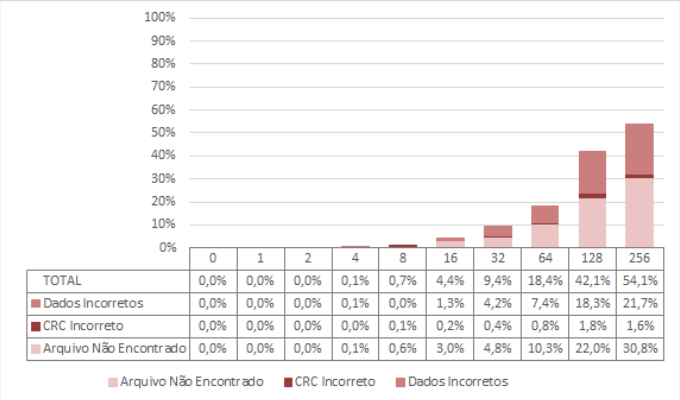
6.2.1. Injeção de falhas na memória SRAM destinada ao *firmware*

O *firmware* não tolerante a falhas apresentou uma média de incidência de defeitos diferente de zero (4,9%) com a injeção de uma única falha por ciclo de leitura (Figura 4). Com a injeção de duas falhas por ciclo de leitura, o número de defeitos foi em média 6,2%, chegando a 99,9% com a injeção de 256 falhas. Em contrapartida, os resultados mostrados na Figura 5 indicam que o *firmware* tolerante a falhas apresentou menos de um por cento de defeitos (0,7%) com injeção de até 8 falhas por ciclo de leitura, sendo que nos testes com até duas falhas injetadas, os defeitos não ocorreram. O primeiro defeito só ocorreu a partir de 4 falhas injetadas, e mesmo assim, a incidência de defeitos foi pequena (menos de 0,1%). Os defeitos somente apresentaram valores significativos a partir da injeção de 16 (4,4%) e 32 falhas (9,6%).



	0	1	2	4	8	16	32	64	128	256
TOTAL	0,0%	4,9%	6,2%	9,7%	14,5%	19,7%	36,1%	60,1%	83,8%	99,9%
Dados Incorretos	0,0%	4,0%	4,6%	6,6%	10,4%	13,2%	24,4%	43,9%	52,7%	26,9%
CRC Incorreto	0,0%	0,1%	0,0%	0,2%	0,4%	0,8%	0,9%	0,8%	2,1%	0,2%
Arquivo Não Encontrado	0,0%	0,8%	1,6%	2,9%	3,7%	5,7%	10,8%	15,4%	29,1%	72,8%

Figura 4. Resultados dos testes com o *firmware* sem técnicas de tolerância a falhas, com a injeção de falhas na região de memória SRAM destinada ao *firmware*.



	0	1	2	4	8	16	32	64	128	256
TOTAL	0,0%	0,0%	0,0%	0,1%	0,7%	4,4%	9,4%	18,4%	42,1%	54,1%
Dados Incorretos	0,0%	0,0%	0,0%	0,1%	0,0%	1,3%	4,2%	7,4%	18,3%	21,7%
CRC Incorreto	0,0%	0,0%	0,0%	0,0%	0,1%	0,2%	0,4%	0,8%	1,8%	1,6%
Arquivo Não Encontrado	0,0%	0,0%	0,0%	0,1%	0,6%	3,0%	4,8%	10,3%	22,0%	30,8%

Figura 5. Resultados dos testes com o *firmware* tolerante a falhas, com a injeção de falhas na região de memória SRAM destinada ao *firmware*.

Com 32 e 64 falhas injetadas, o número de defeitos no *firmware* tolerante a falhas se manteve entre três a quatro vezes menor, e nos casos mais extremos (128 e 256 falhas), o número de defeitos manteve-se aproximadamente pela metade. O número de falhas injetadas por ciclo de leitura limitou-se em 256 porque a partir desse valor, o *firmware* sem tolerância travava completamente em todos os testes, não respondendo sequer ao comando de *reset* do *firmware* monitor, não havendo, portanto, possibilidade de comparação.

Com o *firmware* sem tolerância a falhas, o tipo de defeito mais frequente foi o de dados incorretos, com exceção dos testes com 128 e 256 falhas injetadas. Isso ocorreu porque esses testes, com quantidades excessivas de falhas injetadas, provocaram constantemente o travamento da estação, não gerando o arquivo binário com os dados das leituras. Os defeitos de CRC incorreto foram encontrados poucas vezes, possivelmente porque os dados usados para se gerar o código CRC são mantidos por pouco tempo na memória.

Para chegar a estes resultados, a política de redundância[2] e o algoritmo de votação[3] passaram por diversas melhorias, uma vez que a causa dos problemas eram constantemente avaliadas durante a realização dos testes.

6.2.2. Injeção de falhas em toda memória SRAM

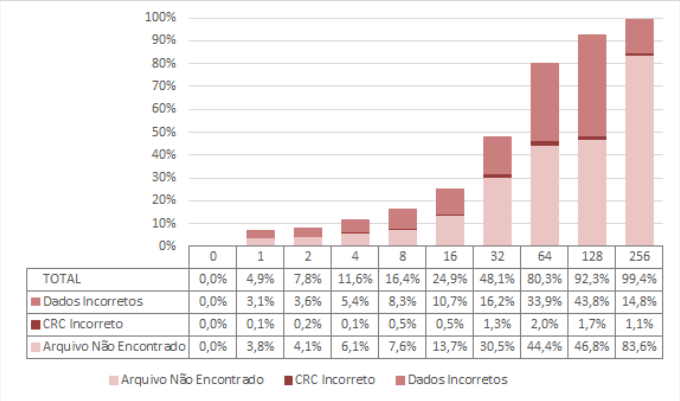
Nos testes com injeção de falhas em ambas as regiões de memória, cujos resultados são mostrados na Figura 6 e na Figura 7, os defeitos mais frequentes foram os do tipo "arquivo não encontrado", pois as falhas injetadas na região de memória SRAM destinada aos controladores internos do microcontrolador provocaram constantemente o travamento do dispositivo. Nenhuma estratégia de redundância foi utilizada para tolerar falhas na região de memória destinada aos controladores devido a grande complexidade que essa implementação envolve. Os testes com o *firmware* tolerante a falhas, em média, apresentaram menos erros em todas as categorias. No teste com 32 falhas injetadas, por exemplo, o número de defeitos do tipo "arquivo não encontrado" foi 30,5% no *firmware* sem tolerância e 23,2% no tolerante. O motivo disso é que falhas injetadas na SRAM destinada ao *firmware* também geram defeitos do tipo "arquivo não encontrado", portanto, o emprego de redundância nessa memória contribuiu para essa redução. Os defeitos de "dados incorretos" também foram menores no *firmware* tolerante a falhas. Com a injeção de 32 falhas por ciclo de leitura, o número de defeitos do sistema não tolerante foi de 16,2% e no tolerante foi de 1,7%.

Comparando os resultados do *firmware* sem tolerância, com injeção apenas na SRAM destinada ao *firmware* e em ambas as memórias SRAM, verificou-se que a ocorrência de defeitos diminuiu de 24,4% para 16,2%. Isso ocorre porque a probabilidade das falhas atingirem os dados do programa diminuiu pela metade, devido ao aumento das

[2]As primeiras versões do *firmware* tolerante a frequentemente apresentavam defeitos do tipo "arquivo não encontrado", mesmo com pequenas quantidade de falhas injetadas. Então, identificamos que a redundância de dados deveria ser implementada não somente nas regiões que armazenavam os dados das leituras, mas também nos endereços de memória que armazenavam variáveis de configuração do programa, como por exemplo, o intervalo de tempo entre as leituras, a quantidade de sensores, etc.

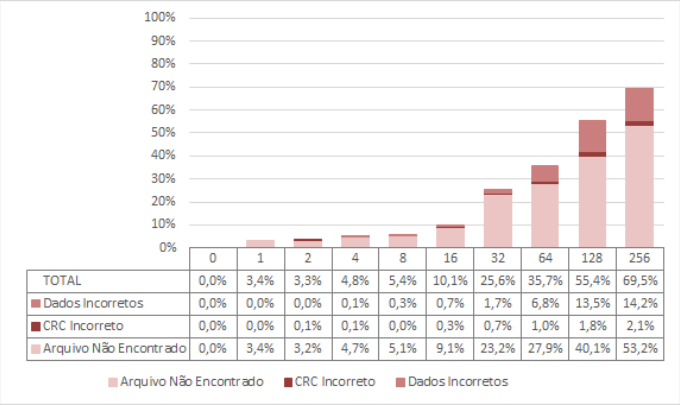
[3]Nos primeiros testes o sistema de votação apenas obtinha o resultado conforme a maioria. Porém, com a injeção de falhas, ao longo do tempo todas as cópias acabavam sendo afetadas, gerando muitos erros. A modificação realizada consistiu em corrigir a cópia com falha, caso possível, após a votação, deixando todas as cópias novamente com o mesmo valor ao final da votação.

regiões para injeção de falhas (de 32 kB para 64 kB).



	0	1	2	4	8	16	32	64	128	256
TOTAL	0,0%	4,9%	7,8%	11,6%	16,4%	24,9%	48,1%	80,3%	92,3%	99,4%
Dados Incorretos	0,0%	3,1%	3,6%	5,4%	8,3%	10,7%	16,2%	33,9%	43,8%	14,8%
CRC Incorreto	0,0%	0,1%	0,2%	0,1%	0,5%	0,5%	1,3%	2,0%	1,7%	1,1%
Arquivo Não Encontrado	0,0%	3,8%	4,1%	6,1%	7,6%	13,7%	30,5%	44,4%	46,8%	83,6%

Figura 6. Resultados dos testes com o *firmware* sem técnicas de tolerância a falhas, com a injeção de falhas na memória SRAM destinada ao *firmware* e na memória SRAM destinada aos controladores internos.



	0	1	2	4	8	16	32	64	128	256
TOTAL	0,0%	3,4%	3,3%	4,8%	5,4%	10,1%	25,6%	35,7%	55,4%	69,5%
Dados Incorretos	0,0%	0,0%	0,0%	0,1%	0,3%	0,7%	1,7%	6,8%	13,5%	14,2%
CRC Incorreto	0,0%	0,0%	0,1%	0,1%	0,0%	0,3%	0,7%	1,0%	1,8%	2,1%
Arquivo Não Encontrado	0,0%	3,4%	3,2%	4,7%	5,1%	9,1%	23,2%	27,9%	40,1%	53,2%

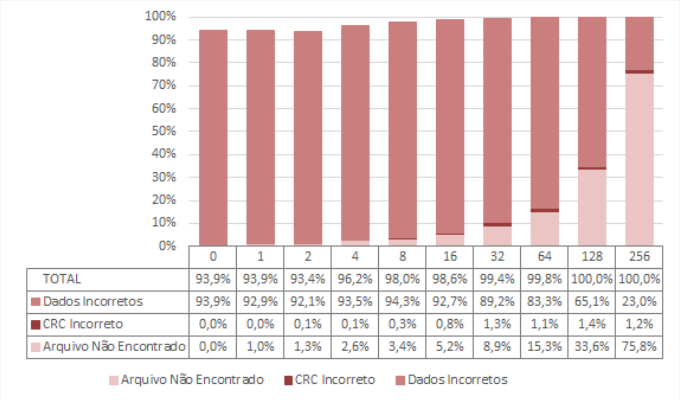
Figura 7. Resultados dos testes com o *firmware* tolerante a falhas, com a injeção de falhas na memória SRAM destinada ao *firmware* e na memória SRAM destinada aos controladores internos.

6.3. Injeção de falhas nas leituras dos sensores e nas regiões de memória SRAM

Os testes que injetaram falhas nas duas regiões de memória SRAM do microcontrolador e nas leituras dos sensores, em geral, mostraram um aumento excessivo dos defeitos do tipo "dados incorretos" no *firmware* sem tolerância a falhas. A injeção de falhas utilizada nos sensores foi de 25% das leituras. Na Seção 6.3.1 exibe-se os resultados para injeção de falhas na SRAM destinada ao *firmware*, e na Seção 6.3.2, os resultados para injeção de falhas em ambas as memórias SRAM. Cada valor apresentado novamente corresponde a média de um conjunto de 30 testes.

6.3.1. Injeção de falhas na SRAM destinada ao *firmware*

Conforme é mostrado na Figura 8, as injeções de falhas nas leituras dos sensores causaram grande quantidade de defeitos no *firmware* sem tolerância a falhas. Comparado aos resultados exibidos na Figura 4, quando se injetou falhas exclusivamente na SRAM destinada



	0	1	2	4	8	16	32	64	128	256
TOTAL	93,9%	93,9%	93,4%	96,2%	98,0%	98,6%	99,4%	99,8%	100,0%	100,0%
Dados Incorretos	93,9%	92,9%	92,1%	93,5%	94,3%	92,7%	89,2%	83,3%	65,1%	23,0%
CRC Incorreto	0,0%	0,0%	0,1%	0,1%	0,3%	0,8%	1,3%	1,1%	1,4%	1,2%
Arquivo Não Encontrado	0,0%	1,0%	1,3%	2,6%	3,4%	5,2%	8,9%	15,3%	33,6%	75,8%

Figura 8. Resultados dos testes com o *firmware* sem técnicas de tolerância a falhas, com a injeção de falhas nas leituras dos sensores e na SRAM destinada ao firmware.

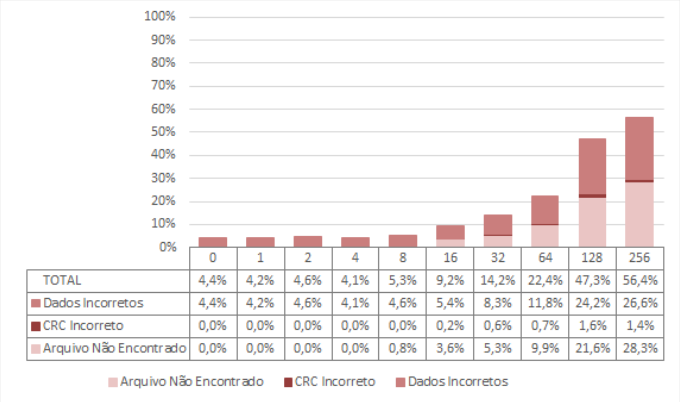
ao *firmware*, verifica-se que a injeção de falhas nas leituras dos sensores aumentou significativamente o número de defeitos. No teste com apenas uma falha injetada, o número de defeitos aumentou de 4,9% para 93,9%. Todos os resultados mostrados neste gráfico apresentam uma incidência de defeitos acima de 93%, sendo o defeito mais comum o do tipo "dados incorretos". Cabe observar que quanto maior o número de falhas injetadas na memória, menor foi a quantidade de defeitos do tipo "dados incorretos". Isso ocorre porque as falhas na memória começaram a travar ou causar um funcionamento incorreto do programa, provocando mais defeitos do tipo "arquivo não encontrado". No teste que injetou 256 falhas por ciclo de leitura, por exemplo, o número de defeitos do tipo "arquivo não encontrado" foi de 75% e o de "dados incorretos", 23%.

Na Figura 9 mostra-se os resultados dos testes com o *firmware* tolerante a falhas. Todos os testes com o *firmware* tolerante a falhas apresentaram uma menor ocorrência de defeitos, comparado ao *firmware* sem tolerância a falhas. No teste com uma única falha injetada na memória e 25% de falhas nas leituras dos sensores, o *firmware* tolerante apresentou 4,2% de defeitos, enquanto o *firmware* não tolerante apresentou 93,9%. O tipo de defeito mais evitado foi o de "dados incorretos", resultado da ação do método *avg* com a redundância de processamento. Esses resultados, se comparados aos da Figura 5, quando se utilizou o mesmo *firmware* e se injetou falhas apenas na SRAM destinada ao *firmware*, mostram ocorrências de defeitos muito próximas, exceto para "dados incorretos", que foram superiores com a injeção de falhas também nos sensores.

6.3.2. Injeção de falhas em ambas as memórias SRAM

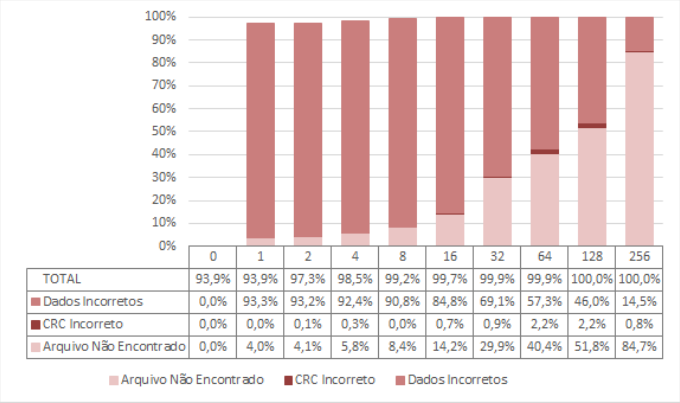
Na Figura 10 e na Figura 11 são exibidos os resultados dos testes com a injeção de falhas nas leituras dos sensores, em conjunto com a injeção de falhas em ambas as memórias SRAM.

Os resultados exibidos na Figura 10 e na Figura 11 tiveram um comportamento análogo aos testes realizados na Seção 6.2.2. Mais uma vez, nos testes com o *firmware* sem tolerância a falhas, os defeitos do tipo "dados incorretos" mantiveram-se acima de 90% nos testes com até 8 falhas injetadas nas regiões de memória SRAM, sendo suces-



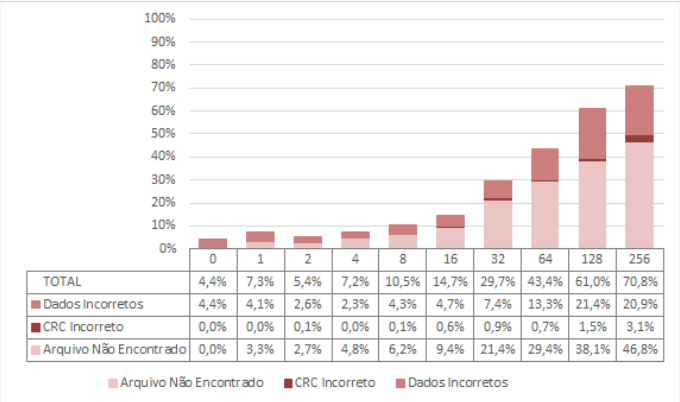
	0	1	2	4	8	16	32	64	128	256
TOTAL	4,4%	4,2%	4,6%	4,1%	5,3%	9,2%	14,2%	22,4%	47,3%	56,4%
Dados Incorretos	4,4%	4,2%	4,6%	4,1%	4,6%	5,4%	8,3%	11,8%	24,2%	26,6%
CRC Incorreto	0,0%	0,0%	0,0%	0,0%	0,0%	0,2%	0,6%	0,7%	1,6%	1,4%
Arquivo Não Encontrado	0,0%	0,0%	0,0%	0,0%	0,8%	3,6%	5,3%	9,9%	21,6%	28,3%

Figura 9. Resultados dos testes com o *firmware* tolerante a falhas, com a injeção de falhas nas leituras dos sensores e na SRAM destinada ao *firmware*.



	0	1	2	4	8	16	32	64	128	256
TOTAL	93,9%	93,9%	97,3%	98,5%	99,2%	99,7%	99,9%	99,9%	100,0%	100,0%
Dados Incorretos	0,0%	93,3%	93,2%	92,4%	90,8%	84,8%	69,1%	57,3%	46,0%	14,5%
CRC Incorreto	0,0%	0,0%	0,1%	0,3%	0,0%	0,7%	0,9%	2,2%	2,2%	0,8%
Arquivo Não Encontrado	0,0%	4,0%	4,1%	5,8%	8,4%	14,2%	29,9%	40,4%	51,8%	84,7%

Figura 10. Resultados dos testes com o *firmware* sem técnicas de tolerância a falhas, com a injeção de falhas nas leituras dos sensores e nas memórias SRAM destinadas ao *firmware* e aos controladores internos do microcontrolador.



	0	1	2	4	8	16	32	64	128	256
TOTAL	4,4%	7,3%	5,4%	7,2%	10,5%	14,7%	29,7%	43,4%	61,0%	70,8%
Dados Incorretos	4,4%	4,1%	2,6%	2,3%	4,3%	4,7%	7,4%	13,3%	21,4%	20,9%
CRC Incorreto	0,0%	0,0%	0,1%	0,0%	0,1%	0,6%	0,9%	0,7%	1,5%	3,1%
Arquivo Não Encontrado	0,0%	3,3%	2,7%	4,8%	6,2%	9,4%	21,4%	29,4%	38,1%	46,8%

Figura 11. Resultados dos testes com o *firmware* tolerante a falhas, com a injeção de falhas nas leituras dos sensores e nas memórias SRAM destinadas ao *firmware* e aos controladores internos do microcontrolador.

sivamente reduzidos a medida que aumentaram as falhas injetadas na memória. A razão do número de defeitos do tipo “arquivo não encontrado” ter ultrapassado o de “dados

incorretos" quando se injetou 128 falhas é que, como já ocorreu em testes anteriores, o microcontrolador travou mais rapidamente devido a inexistência de redundância de dados na SRAM destinada aos controladores internos do microcontrolador. E nesse teste se injetou falhas nesta região.

No *firmware* tolerante a falhas e as mesmas 8 falhas injetadas em ambas as memórias, a ocorrência de defeitos manteve-se abaixo de 11%. Cabe destacar que enquanto o *firmware* não tolerante apresentou mais de 97% de defeitos a partir de duas falhas injetadas por ciclo, o *firmware* tolerante a falhas, com 256 falhas injetadas por ciclo, teve em média 70% de defeitos.

6.4. Consumo de energia

O consumo de energia da estação meteorológica é maior durante o período de leitura dos sensores devido aos circuitos eletrônicos auxiliares que precisam ser acionados. Além disso, a carga de processamento aumenta nesse período. Depois que os dados lidos são gravados na memória flash, a estação entra em um período inativo e o consumo diminui consideravelmente, até o próximo ciclo. As medições do consumo de energia mostraram que durante o período de leitura dos sensores, a energia consumida subiu de 1,08J (valor médio de 30 medições) no *firmware* sem tolerâcia a falhas, para 3,26J (valor médio de 30 medições) no *firmware* tolerante a falhas. A energia consumida pelo segundo foi superior devido ao maior tempo de processamento durante a leitura dos sensores.

Na Figura 12 mostra-se a energia total consumida (energia consumida para ler/-processar sensores mais energia em modo inativo) em um ciclo de leitura com duração de 10 segundos. Com o *firmware* sem tolerância a falhas, o consumo médio apresentado foi de 5,63J, e com o *firmware* tolerante a falhas, o consumo foi de 5,88J. Em uma situação real de utilização, o tempo de inatividade de uma estação meteorológica é maior, pois os dados são coletados em intervalos de minutos, e não de segundos. Logo, o impacto no consumo de energia é ainda menor.

6.5. Comparação do consumo de energia, memória *flash*, memória SRAM e tempo de processamento

Conforme visto da Seção 6.1 à Seção 6.3, as técnicas de tolerância a falhas aumentaram a segurança do sistema, pois o número de defeitos gerados com a presença de falhas diminuiu. Porém, as alterações (principalmente a redundância de processamento) afetaram o tempo total de processamento. Na Tabela 2 pode-se conferir o tempo necessário para se ler todos os sensores e gravar os dados das leituras em um arquivo na memória *flash*, para estação meteorológica sem tolerância a falhas; com tolerância, mas sem injeção de falhas; e, com tolerância e injeção de falhas (um caso médio e um extremo).

Na Tabela 3 é possível observar que as técnicas de tolerância a falhas aumentaram também o consumo de memória *flash*, o consumo de memória RAM e o consumo de energia.

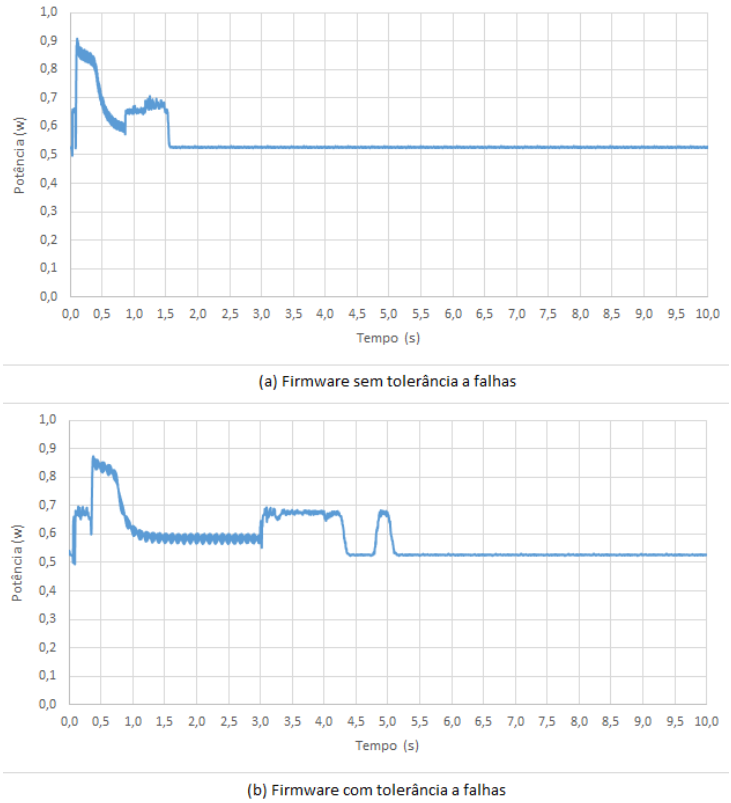


Figura 12. Potência da estação ao longo do tempo para o *firmware* sem tolerância a falhas e com tolerante a falhas.

Tolerância a falhas	Número de falhas injetadas na memória	Porcentagem de falhas injetadas nas leituras dos sensores	Tempo médio de processamento
Não	0	0%	1,78s
Sim	0	0%	5,03s
Sim	16	25%	5,76s
Sim	128	50%	8,62s

Tabela 2. Tempo médio de processamento.

Tolerância a falhas	Consumo (energia)	Consumo (*flash*)	Consumo (RAM)	Tempo de processamento
Não	33,78J	39,6 kB	1,7 kB	1,78s
Sim	35,28J	68,4 kB	3,9 kB	5,03s

Tabela 3. Comparação do consumo de energia (durante um minuto), memória flash, memória RAM e tempo de processamento entre os *firmwares* sem tolerância a falhas e com tolerância a falhas.

7. Conclusão

Programas sem técnicas de tolerância a falhas podem facilmente travar ou produzir resultados incorretos (defeitos) quando expostos a situações de falhas. O uso das técnicas

baseadas na redundância de dados e de processamento, perfeitamente viáveis para sistemas embarcados de baixo custo, trouxeram uma melhoria significativa na segurança do sistema usado como estudo de caso neste trabalho. Nos testes realizados, a ocorrência de defeitos na estação meteorológica diminuiu drasticamente, especialmente nos testes que realizaram a injeção de falhas nas leituras dos sensores. Nesses testes, com uma taxa de 25% de injeção de falhas nas leituras dos sensores, a ocorrência de defeitos foi em média de 93,95% no *firmware* não tolerante a falhas e apenas 4,38% no *firmware* tolerante a falhas.

Nos testes em que as falhas foram injetadas em ambas as regiões de memória os resultados foram menos satisfatórios. Isso demonstra ainda mais a importância das técnicas de tolerância a falhas, uma vez que na memória SRAM destinada aos controladores internos do microcontrolador, não foi utilizado nenhum tipo de redundância, sendo esta, a área de maior vulnerabilidade do sistema.

Referências

Chetan, S., Ranganathan, A., and Campbell, R. H. (2005). Towards fault tolerance pervasive computing. *IEEE Technol. Soc. Mag.*, 24(1):38–44.

Dubrova, E. (2007). Fault tolerant design: An introduction. Department of Microelectronics and Information Technology Royal Institute of Technology Stockholm, Sweden. Stockholm, Sweden: Kluwer Academic Publishers.

Huang, B., Li, X., Li, M., Bernstein, J., and Smidts, C. (2005). Study of the impact of hardware fault on software reliability. *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, 0:63–72.

Kruger, K. and Iaione, F. (2013). Development of a fault injection system to test a weather station based on rapid prototyping platform. In *IEEE International Conference*, pages 1652–1657. High Performance Computing and Communications & Embedded and Ubiquitous Computing.

Laprie, J.-C., Béounes, C., and Kanoun, K. (1990). Definition and analysis of hardware- and software-fault-tolerant architectures. *Computer*, 23(7):39–51.

mbed (2014). The mbed official page.

Nelson, V. P. (1990). Fault-tolerant computing: Fundamental concepts. *Computer*, 23:19–25.

Patterson, D. A. and Hennessy, J. L. (2005). *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition.

Singh, A. D. and Murugesan, S. (1990). Fault-tolerant systems. *Computer*, 23:15–17.

Velazco, R. and Rezgui, S. (2000). Transient bitflip injection in microprocessor embedded applications. In *Proceedings of the 6th IEEE International On-Line Testing Workshop (IOLTW)*, IOLTW '00, pages 80–, Washington, DC, USA. IEEE Computer Society.