SISTEMAS TOLERANTES A FALHAS

FAULT TOLERANT SYSTEMS

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E SISTEMAS DIGITAIS - PCS

GRUPO DE ANÁLISE DE SEGURANÇA - GAS

PROF. JOÃO BATISTA CAMARGO JR.

2020

Sumário

1. Apresentação

Um sistema tolerante a defeito deve atender aos requisitos:

- Confiabilidade;
- Disponibilidade;
- Segurança;
- Desempenho;
- Dependabilidade;
- Manutenabilidade;
- Testabilidade.

Um sistema tolerante a defeito é aquele que continua desempenhado corretamente suas tarefas específicas na presença de defeito de hardware e de software.

A tolerância a defeito é um ATRIBUTO que permite ao sistema alcançar a "operação tolerante a defeito".

Definições:

- **Confiabilidade** ("Realiability"):R(t)

O conceito de confiabilidade corresponde à probabilidade que um determinado sistema irá operar corretamente, ou seja, de acordo com sua especificação de requisitos, durante um determinado intervalo completo de tempo.

Desta forma, *R(t)* é a probabilidade condicional que o sistema irá desempenhar corretamente no intervalo de tempo *[to,t]*, dado que o sistema estava desempenhado corretamente no instante to. Trata-se de uma probabilidade condicional, pois ela depende do sistema estar operacional no início deste intervalo de tempo escolhido.

De forma complementar, pode-se definir o conceito de Não-confiabilidade

A Não-Confiabilidade ("Unreliability") - *Q (t)* é definida como a probabilidade que um sistema irá realizar as tarefas de maneira incorreta, ou seja, desrespeitando sua especificação de requisitos, durante um intervalo de tempo *[to,t]* , dado que o sistema estava desempenhando corretamente no instante to. Este conceito também é conhecido como Probabilidade de Falha.

O conceito de Tolerância a Falha corresponde a uma técnica que pode aumentar a Confiabilidade de um sistema. No entanto, um sistema, com

características de tolerância a defeito, não apresenta, necessariamente, uma alta confiabilidade. Da mesma forma, pode-se afirma que, um sistema, com alta confiabilidade, não necessariamente possui características de tolerância a defeito.

- **Disponibilidade (Availability - A (t)).**

O conceito de disponibilidade corresponde à probabilidade que um determinado sistema esteja operando corretamente e disponível para realizar suas funções num instante de tempo *t.* Assim, a disponibilidade difere da confiabilidade de forma que, a confiabilidade depende de um *INTERVALO* de tempo enquanto que a disponibilidade é avaliada em um *INSTANTE* de tempo.

A disponibilidade depende então não apenas o quão freqüente um sistema torna-se inoperante, mas também o quão rápido o sistema pode ser reparado.

Num caso prático, um processador reserva pode desempenhar as funções do sistema enquanto que o processador principal está sendo reparado, conservando desta forma o sistema disponível para uso final.

- **Segurança (Safety- S (t)).**

É a probabilidade que um sistema irá desempenhar, num intervalo de tempo, suas funções corretamente ou descontinuar suas funções de uma forma que não interrompa a operação de outros sistemas de segurança nem comprometa a segurança de pessoas, dado que esteja funcionando no instante inicial. A segurança é uma média da capacidade do sistema ser “Fail- Safe”.

Se o sistema não opera corretamente, você quer que ele falhe, pelo menos, de uma maneira segura.

- **Desempenho - (“ Performability - P (L, t)”).**

Em muitos casos é possível projetar sistemas que podem continuar a realizar as funções corretamente após falhas de HW e erros no SW, mas não no mesmo nível de desempenho.

- **Definição:**

Probabilidade que o desempenho de um sistema estará maior ou igual a um nível **L,** no instante **t.** É a média de probabilidade que um subconjunto de funções está sendo realizado corretamente.

- **Manutenabilidade (“Maintainability -M (t)”).**

É a facilidade com a qual um sistema pode ser reparado, uma vez que ele falhou.

É a probabilidade que um sistema em falha será reparado para o estado operacional dentro de um **período de tempo t.**

Muitas técnicas de tolerância a defeito podem ser utilizadas para detectar e localizar problemas no sistema, com o propósito de manutenção.

Os diagnósticos automáticos podem auxiliar muito neste aspecto.

- Testabilidade (“ Testability”).

É a habilidade de testar certos atributos dentro de um sistema. Mede a facilidade com a qual certos testes podem ser realizados.

Muitas técnicas que são vitais no sentido de se alcançar “tolerância a defeito” podem ser utilizadas para detectar e localizar problemas aumentando a Testabilidade. Está intimamente relacionada com a manutenabilidade, devido ao fato de diminuir o tempo requerido para identificar e localizar o problema.

- Dependabilidade (“Dependability”).

Aglutina, por exemplo, os conceitos de Confiabilidade, Disponibilidade, Segurança, Manutenabilidade, desempenho e testabilidade.

2. EVOLUÇÃO TECNOLÓGICA E APLICAÇÕES

A evolução tecnológica dos Sistemas de Proteção pode ser esquematizada em quatro etapas:

- Sistemas a Relês;
- Sistemas com Tecnologia a Estado Sólido;
- Sistemas CLP redundantes; e
- Sistemas Tolerantes a Defeitos com Redundância Triplicada.

Nos sistemas implementados a Relês tem-se um aumento significativo da sua complexidade á medida que se expande a funcionalidade, tornando, desta forma, a manutenção, a localização de defeitos e prováveis modificações, difíceis de serem realizadas. Convém destacar também os altos custos deste tipo de sistema. Apesar dos relês apresentarem um modo de falha bastante conhecido, este sistemas trabalham apenas com entrada e saídas digitais.

Com relação aos Sistemas Baseados em Lógica do Estado Sólido, estes apresentam boa testabilidade e localização de defeitos. Por outro lado, estes sistemas apresentam uma capacidade limitada de diagnóstico e pouca flexibilidade para modificações, aceitam somente entradas e saídas digitais, além de utilizarem componentes de difícil reposição.

Em função dessas dificuldades e da evolução tecnológica, passou - se a utilizar sistemas com Controladores Lógicos Programáveis - CLP ‘ s redundantes, implementados através de dois processadores em paralelo sendo um deles selecionado através de chaveamento. Esses processadores além de aceitarem entrada e saídas analógicas, melhoraram a capacidade de diagnóstico, podendo ser usados justamente com um “Bus de dados”. Nesses sistemas torna-se difícil definir qual o processador que está correto, o chaveamento entre ambos não é seguro, há a necessidade de duplicação das entradas e saídas dos processadores e alto risco em mudança nas suas programações. Neste sentido, vale também destacar a dificuldade em se verificar os programas contidos nos processadores.

No sentido de superar todas as dificuldades e com o aumento da complexidade dos Sistemas de Proteção, chegou - se aos Sistemas Tolerantes a Defeito com redundância Modular Triplicada. Dentro desta filosofia destacam-se duas

linhas de trabalho, com ênfase em Hardware e Software, sendo a última adotada com mais freqüência.

Alguns dos requisitos desejáveis desta nova arquitetura são:

- uma falha simples no sistema não deve gerar entradas ou saídas erradas nem deve evitar que o sistema funcione conforme projetado;
- qualquer falha deve gerar alarme e uma indicação do local da ocorrência;
- qualquer falha simples deve ser reparada "on -line" sem interrupção da operação do Sistema de Segurança.

Como conseqüência das necessidades do mundo moderno e da complexidade crescente dos Sistemas de proteção exige-se um trabalho de análise bastante amplo na verificação dos Requisitos Gerais de Segurança, no sentido de caminhar-se em direção a uma melhor "Segurança" e "Qualidade" desses sistemas.

Desta forma deve-se ter como principais metas nestes Sistemas de Segurança as seguintes características:

- Alta Disponibilidade;
- Baixo tempo Médio para Reparo ("Mean Time to Repair"- MTTR);
- Alto Tempo Médio entre Falhas ("Mean Time Between Failures"- MTBF);
- Alto Tempo Médio entre Falhas Inseguras ("Mean Time Between Unsafe Failures"- MTBUF);
- Metodologia de Programação que dê ênfase à segurança;
- Diagnóstico Exaustivo;
- Interface Amigável;
- Comunicação Segura via Rede; e
- Excelente Documentação.

A Tolerância a Defeito é atualmente requerida, pois os novos sistemas eletrônicos são menos confiáveis.

3. APLICAÇÕES

As aplicações de computação tolerante a defeitos podem ser categorizadas em 4 áreas:

- Vida longa
- Computação crítica
- Aspecto de manutenção adiantada/adiada
- Alta disponibilidade

3.1. Aplicações de Vida Longa

Sistemas a bordo de naves e satélites.

Requisitos típicos desta aplicação é ter uma probabilidade de 0,95 de estar operacional no fim de um período de 10 anos.

No entanto, podem suportar pequenos períodos fora do ar (1 semana).

3.2. Aplicações críticas

Relacionadas com Vidas Humanas, meio-ambiente, ou proteção de equipamento.

- Controladores de Vôo
- Sistemas militares
- Certos controles industriais
- Controle metrô - Ferroviário, etc...
- Indústria química

Requisito típico: Confiabilidade > 0,97 no fim de um período de três horas, no entanto os requisitos podem variar em função das particularidades do sistema.

3.3. Aplicações com Manutenção Adiantada/ Adiada

Aplicáveis em sistemas que as operações de manutenção são extremamente custosas, inconvenientes ou difíceis de realizar.

Exemplo:

- Estações de processamento remoto. (custo) (Centrais Telefônicas)
- Certas aplicações espaciais (local).

Visitas periódicas que realizam uma manutenção preventiva muito forte.

Evitar manutenção não programada.

Durante os períodos entre visitas, o sistema manipula as falhas e serviços errados de maneira autônoma.

3.4. Aplicações de Alta Disponibilidade

Bons exemplos: Sistemas Bancários e “Time-Shared"

Os usuários destes sistemas requerem ter uma probabilidade alta de receber o serviço quando requisitado.

4. Definições Fundamentais

Algumas definições fundamentais na área de confiabilidade e disponibilidade devem ser formalizadas visando um maior esclarecimento da terminologia sendo utilizada. A figura 4.1 apresenta a relação entre os termos *Defeito*, *Erro* e *Falha*.

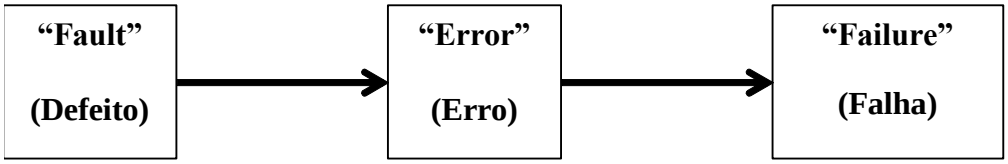


Figura 4.1. Relação entre Defeito, Erro e Falha

O termo *Defeito* corresponde a um problema intrínseco de um componente. Pode corresponder a um *defeito físico* ou *imperfeição*, que ocorre dentro de um componente de hardware ou software.

Como exemplo de *defeito* pode ser citado um condutor aberto ou fechado, uma imperfeição física num dispositivo semicondutor ou, um laço infinito num programa de computador.

O termo *Erro* corresponde à manifestação interna de um *defeito*. Especificamente um *erro* é um desvio da precisão ou correção esperada de uma informação interna ao sistema.

Uma *Falha* ocorre quando o sistema desempenha incorretamente uma de suas funções.

É também denominada como mau funcionamento. Dessa forma, uma *Falha* corresponde também ao desempenho de alguma função numa quantidade ou qualidade abaixo do normal. A *falha* é uma exteriorização do *erro*.

Seja o exemplo de um Votador dois de três, apresentado na figura 4.2.

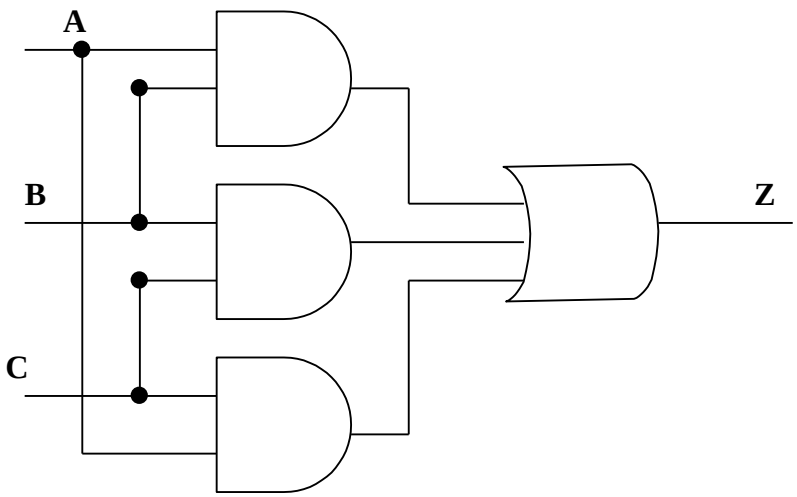


Figura 4.2 – Circuito Votador

A Tabela da Verdade deste circuito está apresentada na tabela 4.1, que se segue.

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Tabela 4.1 – Tabela de Verdade do Circuito Somador

Supondo a ocorrência de um *Defeito* intrínseco no componente Votador, como por exemplo, um curto entre o ponto A e o *Vcc*, a nova Tabela da Verdade passa a apresentar a seguinte constituição apresentada na Tabela 4.2.

A	B	C	Z	
0	0	0	0	
0	0	1	1	(*)
0	1	0	1	(*)
0	1	1	1	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	1	

Tabela 4.2. Nova Tabela da Verdade do Circuito Somador

Pode-se observar que os valores com (*) correspondem às saídas *Z* que contém *erros* na informação interna do sistema sendo considerado. Se a saída *Z* estiver controlando um circuito externo, como por exemplo, um circuito a Relê, toda vez que o dado contaminado de entrada *A* for utilizado, o *erro* na informação *Z* será exteriorizado, transformando-se em uma *Falha*.

O *Defeito* está relacionada com o *Universo Físico*, representado através dos dispositivos semicondutores, fontes, além de outras entidades físicas de hardware e componentes de software.

O *Erro* está relacionado com o *Universo de Informação*, podendo afetar sua precisão ou conteúdo. Os termos aplicáveis a este universo podem incluir *erro de paridade, erro de bit, etc...*

A *Falha* está relacionada com o *Universo Externo*, ou seja, aquele em que o usuário final enxerga o sistema e o seu mau funcionamento.

Com relação aos instantes de ocorrência entre um *Defeito*, um *Erro* e uma *Falha*, pode-se definir o conceito de *Latência*.

A *Latência* de um *Defeito* corresponde ao intervalo de tempo entre a ocorrência de um *Defeito* e o aparecimento do correspondente *Erro*.

A *Latência* de um *Erro* corresponde ao intervalo de tempo entre a ocorrência de um *Erro* e o aparecimento da correspondente *Falha*.

4.1 Causas dos Defeitos

Os *Defeitos* podem ser resultado de uma variedade de eventos que ocorrem internamente a um determinado componente eletrônico ou computacional, externamente ao componente ou durante o projeto do componente ou sistema.

As causas possíveis dos *Defeitos* podem, desta forma, estar associadas com os seguintes aspectos:

- Problema na Especificação;
- Problema na Implementação;
- Desgaste do Componente;
- Fatores Externos.

A figura 4.3. a seguir ilustra a relação entre as causas fundamentais das *Defeitos* e a relação com os *Erros* e *Falhas*.

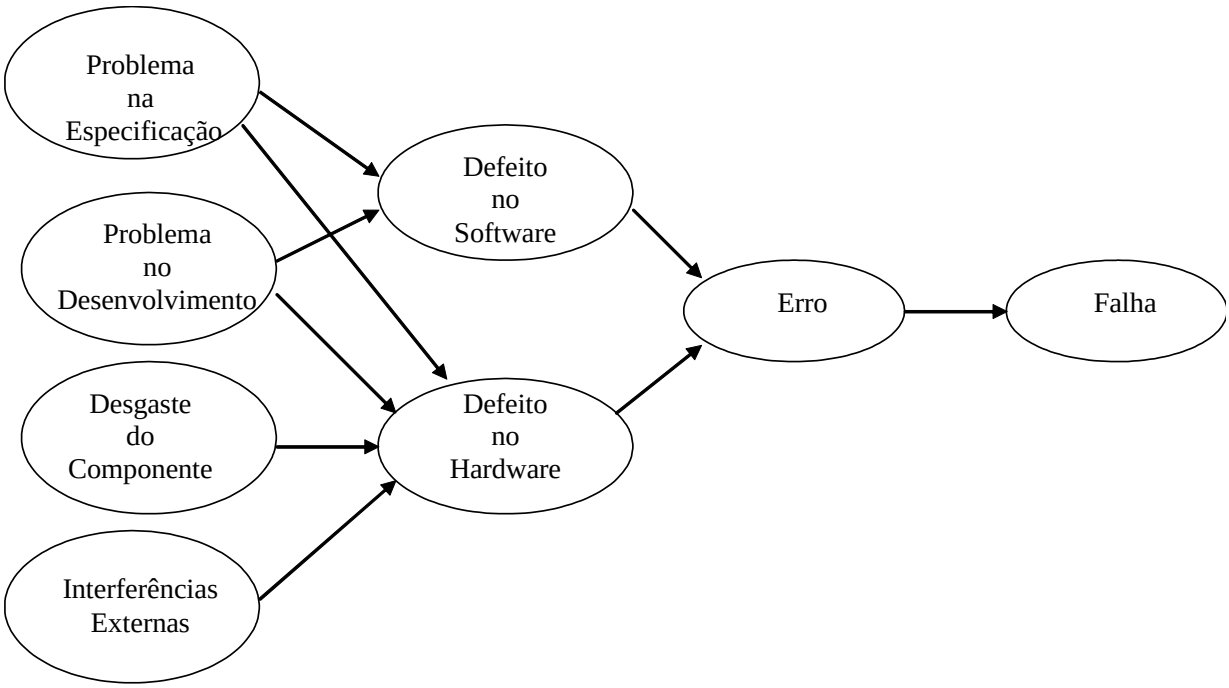


Figura 4.3 – Causas dos Defeitos, Erros e Falhas

Os Defeitos podem ser caracterizadas de acordo com sua Causa, Natureza, Duração, Extensão e Valor. O detalhe desta relação é apresentado de forma esquemática na tabela 4.3.

<table>
<tr><td rowspan="12">Características dos Defeitos</td><td rowspan="4">Causa</td><td colspan="2">Problema na Especificação</td></tr>
<tr><td colspan="2">Problema na Implementação</td></tr>
<tr><td colspan="2">Desgaste</td></tr>
<tr><td colspan="2">Interferências Externas</td></tr>
<tr><td rowspan="3">Natureza</td><td rowspan="2">Hardware</td><td>Analógico</td></tr>
<tr><td>Digital</td></tr>
<tr><td colspan="2">Software</td></tr>
<tr><td rowspan="3">Duração</td><td colspan="2">Permanente</td></tr>
<tr><td colspan="2">Intermitente</td></tr>
<tr><td colspan="2">Transiente</td></tr>
<tr><td rowspan="2">Extensão</td><td colspan="2">Local</td></tr>
<tr><td colspan="2">Global</td></tr>
<tr><td rowspan="2">Valor</td><td colspan="2">Determinado</td></tr>
<tr><td colspan="2">Indeterminado</td></tr>
</table>

Tabela 4.3 – Características dos Defeitos

4.2. Filosofias de Projeto para Combater os Defeitos

Conforme apresentado anteriormente, podem existir *Defeitos*, *Erros* e *Falhas*. Existem técnicas que podem ser utilizadas para evitar os Defeitos, outras para mascarar os Defeitos, e outras para a Tolerância a Defeitos/Erros. Estas técnicas são apresentadas na figura 4.4.

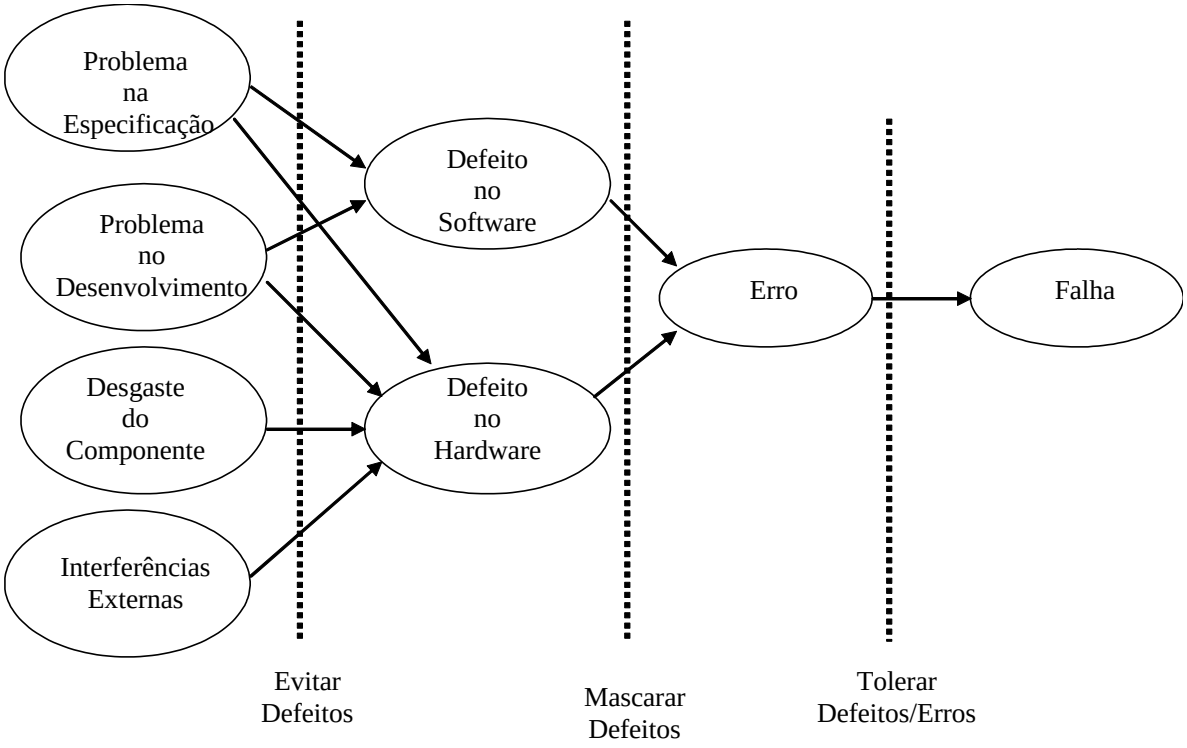


Figura 4.4. – Técnicas para Combater os Defeitos/Erros

Para se Evitar os Defeitos podem ser citadas as seguintes técnicas:

- Revisões de Projeto;
- Testes;
- Métodos de Controle de Qualidade;
- Métodos Preventivos.

Para se Mascarar os Defeitos podem ser realizadas as seguintes ações:

- Correção dos Erros;
- Votação Majoritária, entre outros.

Com relação à técnica de Tolerância a Defeitos/Erros, o sistema deve apresentar a habilidade de continuar desempenhando corretamente suas funções mesmo após a ocorrência da Defeito/Erro. Nesse sentido devem ser tomadas algumas atitudes para que essa tolerância seja alcançada. Podem ser adotadas as seguintes tarefas:

- Mascaramento dos Defeitos
- Reconfiguração do sistema: deve-se detectar e localizar o Defeito/Erro e reconfigurar o sistema visando a remoção do componente em defeito.
    - A detecção do Defeito irá determinar a necessidade das outras ações. Está relacionada com o Fator de Cobertura de Defeitos.
    - A localização do Defeito irá direcionar a forma de recuperação
    - Isolamento do Defeito, prevenindo que seus efeitos se propaguem através do sistema.
- Recuperação do Defeito, fazendo com que o sistema permaneça operacional através de uma ação de reconfiguração.

5. Técnicas de Projeto para Alcançar Tolerância a Defeitos.

O conceito de Redundância

No início o conceito de redundância estava intimamente relacionado com a repetição de bloco de hardware.

De acordo com Johnson: "Redundância é simplesmente a adição de informação, recursos ou tempo além do que é necessário para a operação normal do sistema".

FORMAS DE REDUNDÂNCIA

1. Redundância de Hardware: adição extra de hardware, geralmente com o objetivo de detenção ou tolerância a defeito.
2. Redundância de Software: adição extra de software, além do necessário para realizar uma função para detectar ou possivelmente tolerar defeitos.
3. Redundância de Informação: adição extra de informação além do requerido para implementar uma dada função; ex.: código de detecção de erro.
4. Redundância de tempo: usa tempo adicional para realizar as funções de um sistema de maneira que a detecção e a tolerância a defeitos possam ser alcançadas. (Ex: detectar erros transientes).

O uso de redundância pode fornecer capacidade adicional num sistema. Na realidade, se tolerância a defeito ou detecção de defeito são requeridos, alguma forma de redundância é requerida.

Mas a redundância pode ter um impacto importante no desempenho do sistema, tamanho, peso, consumo de energia, custo e confiabilidade.

5.1. Redundância de Hardware.

Há três tipos básicos de redundância de hardware:

- Passiva: usa o conceito de mascaramento de defeitos para esconder a ocorrência de defeitos e prevenir que defeitos resultem em erros. Não requer nenhuma ação do sistema ou do operador.
- Ativa: conhecido como método dinâmico, alcança tolerância a defeito por detecção de defeitos e realizando alguma ação para remover o hardware com defeito. (reconfiguração).

Detecção de defeitos $\longrightarrow$ localização do defeito $\longrightarrow$ recuperação do defeito.

- Híbrida: combina técnicas passivas e ativas.

5.1.1. Redundância de Hardware Passiva.

Baseia-se no mecanismo de votação (votação majoritária), para mascarar a ocorrência de defeitos.

Redundância de Hardware Passiva mais comum:

“TMR - Triple Modular Redundancy”

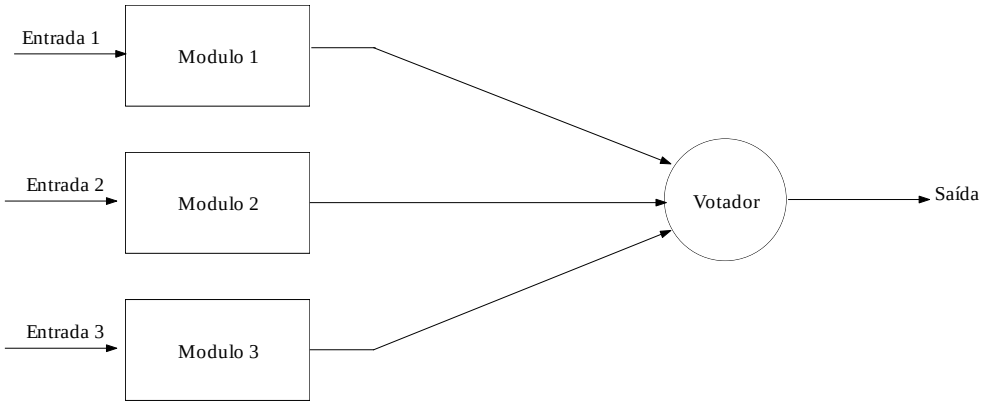


Figura 5.1

O TMR pode ser aplicado á software onde três versões diferentes do programa são usadas para proteger contra defeito no software em uma das três versões.

A maior dificuldade no TMR é o votador; se o votador falha, o sistema completo falha.

Em outras palavras, a confiabilidade do TMR simples não pode ser melhor do que a confiabilidade do votador. (“fail-safe”).

Muitas técnicas podem ser usadas para recuperar a falha no votador.

Uma abordagem é triplicar o votador e fornecer três resultados independentes.

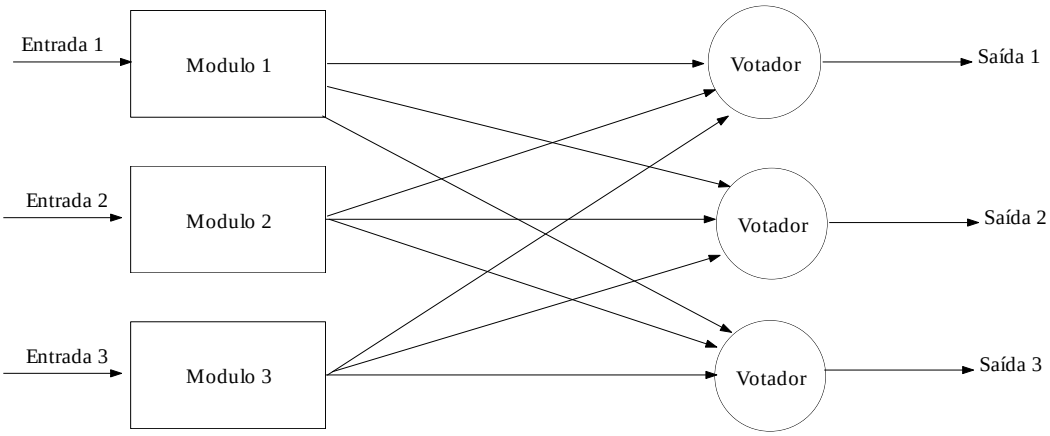


Figura 5.2

Muitos estágios do TMR podem ser interconectados usando a seguinte abordagem:

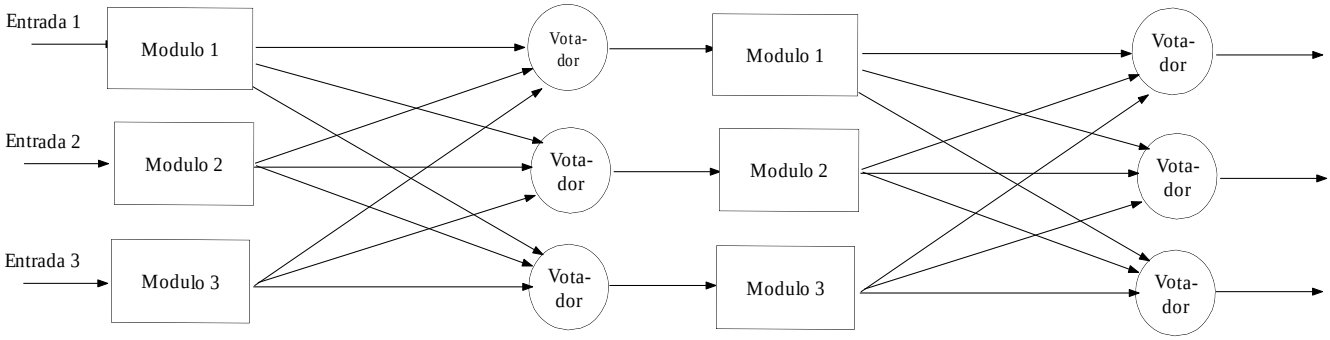


Figura 5.3

Redundância N-Modular

Generalização do TMR. A diferença é que são utilizados N módulos.

Normalmente N é um número impar, para efeitos de votação majoritária.

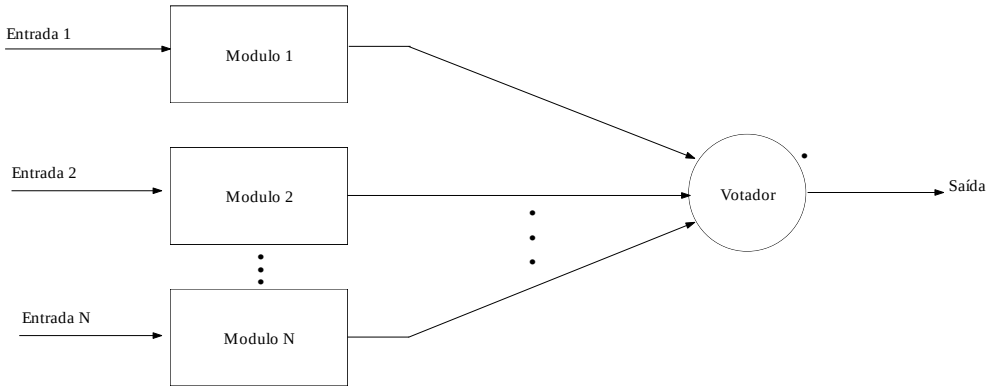


Figura 5.4

Neste caso mais do que um módulo com defeito podem ser tolerados.

Em muitas aplicações críticas, dois defeitos devem ser tolerados para permitir alcançar a confiabilidade requerida e a capacidade de tolerar defeitos.

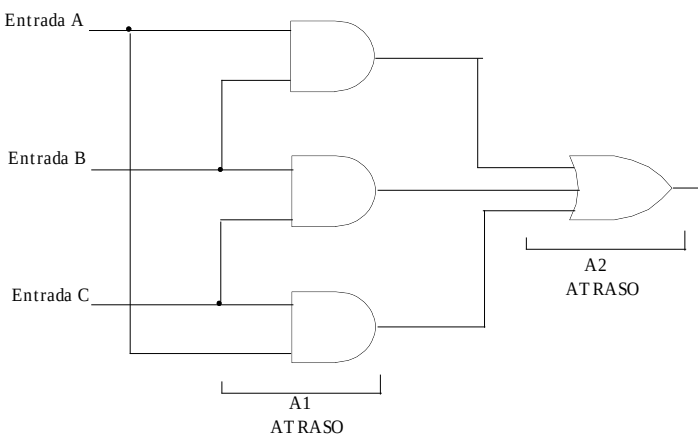
Limitante de N: consumo, peso, custo, confiabilidade do próprio módulo e tamanho.

Técnicas de votação

A votação não apenas engloba decisões de projeto (em que instante votar), mas também impõe muitas considerações como, votação por hardware ou por software.

Votador em hardware para dados digitais são relativamente simples de projetar.

O circuito a seguir implementa um votador majoritário



A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figura 5.5

Na maioria das aplicações práticas, o “timing” é um aspecto crítico no procedimento de votação. Se os valores chegam no votador em tempos poucos diferentes, resultados incorretos podem ser gerados temporariamente. Para contornar estes problemas, flip-flop’s podem ser utilizados nas entradas dos votadores para sincronizar o processo de votação.

O circuito a seguir procura resolver este problema.

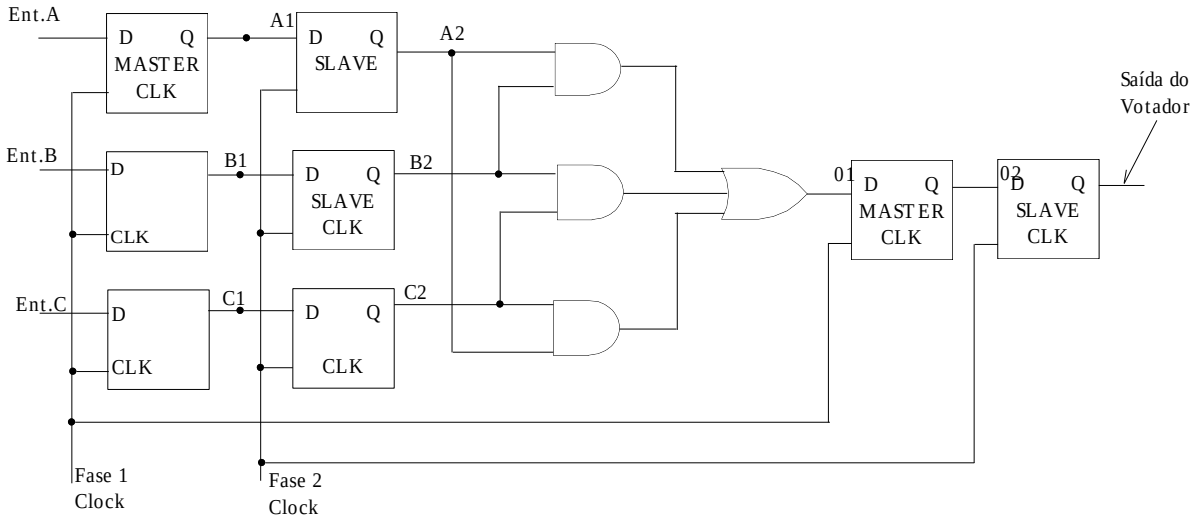


Figura 5.6

CARTA DO TEMPO

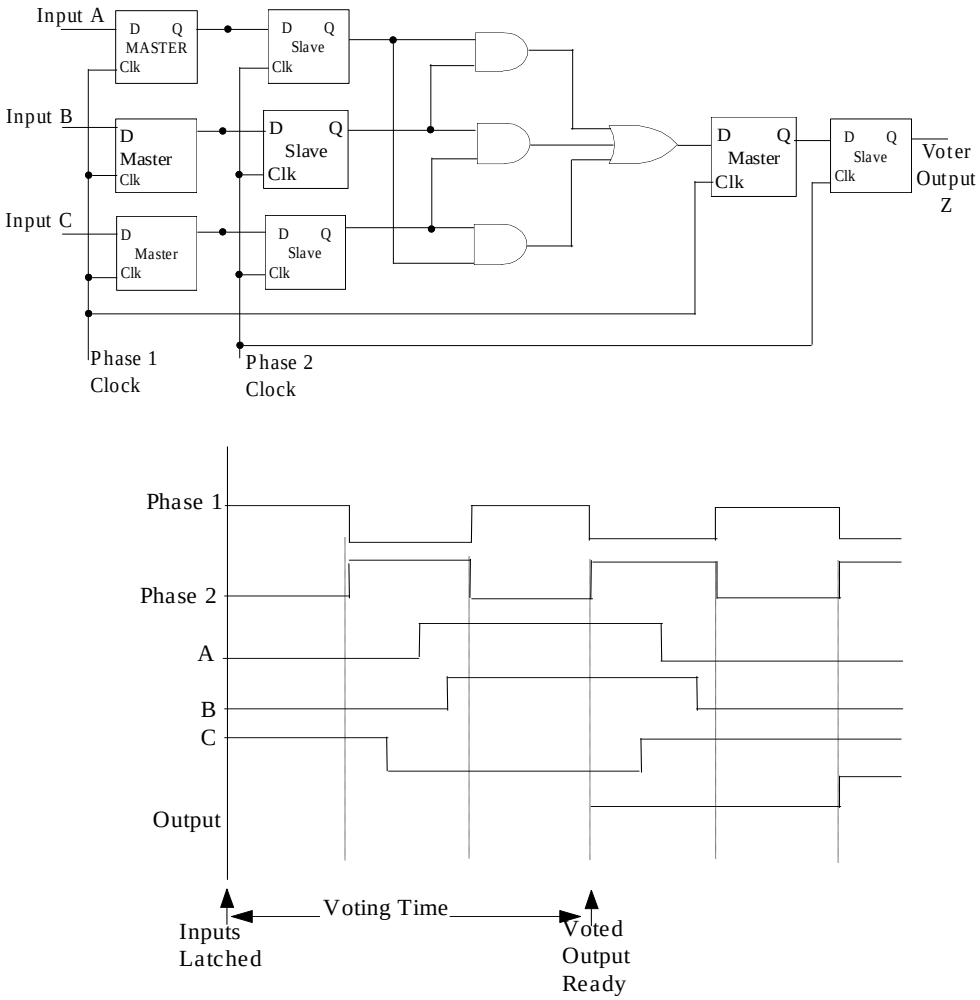


Figura 5.7

Quando se tem votação por hardware:

- atraso depende dos atrasos das pastilhas/componentes usados para construir o circuito.
- Número de componente bastante grande aumento do consumo, peso, tamanho.

Quando se tem votação por software:

- mínimo hardware;
- simples modificar o software;
- pode requerer mais tempo que um hardware dedicado.

As decisões pela votação em hardware ou software devem levar em considerações os seguintes aspectos:

1) Disponibilidade do processador para realizar a votação
2) Velocidade que a votação deve ser realizada
3) Criticidade quanto a espaço, potência e peso.
4) número total de votadores
5) flexibilidade para futuras mudanças.

Outro problema em votação é que os valores de uma votação podem não concordar perfeitamente devido a precisão e tolerância.

Desta forma, uma saída é através da técnica "Mid-Value" (entre o valor correto sem erro e o com erro em votação com número ímpar de módulos).

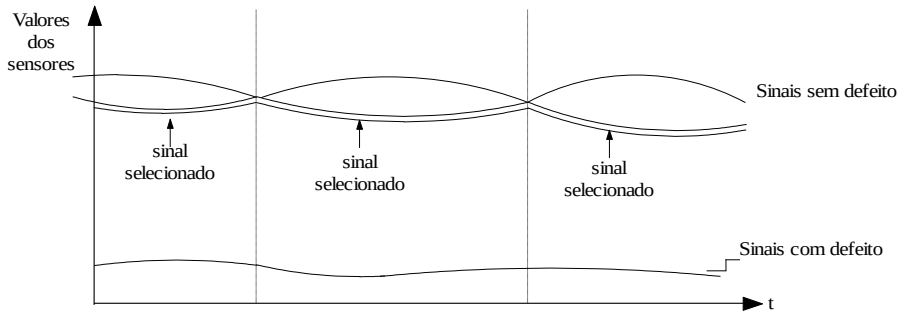


Figura 5.8

Outra maneira é ignorar os últimos bits menos significativas da informação em função da precisão dos componentes sendo utilizados. Em outras palavras, a votação é realizada sobre os R bits mais significativos.

5.1.2. Redundância de Hardware Ativa.

Esta técnica atinge á tolerância a defeito através da detecção do defeito, localização do defeito e recuperação do defeito. (detecção do erro, localização do erro e recuperação do erro).

Neste tipo de redundância o erro é tolerado temporariamente até que o sistema seja reconfigurado.

Exemplos dessas técnicas:

a) Duplicação com Comparação

Este tipo de técnica é apresentado na figura a seguir:

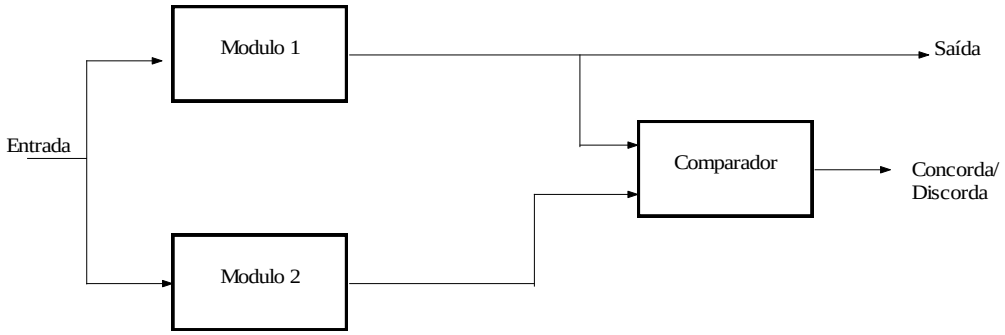


Figura 5.9

Cada módulo realiza as mesmas funções em paralelo. Se ocorrer uma discordância, mensagem de erro é gerada.

Na maioria dos casos é detectada a existência do defeito, mas não é tolerado, pois não há meios de saber qual módulo está com defeito. Mas é um meio de se fazer detecção de defeito.

Problemas:

- Entradas com defeito, resultados errados em ambos os módulos;
- Comparador, pode não executar sua função exatamente (precisão);
- Falha no Comparador: não indicar erros quando tem.

A técnica que resolve alguns dos problemas anteriores é:

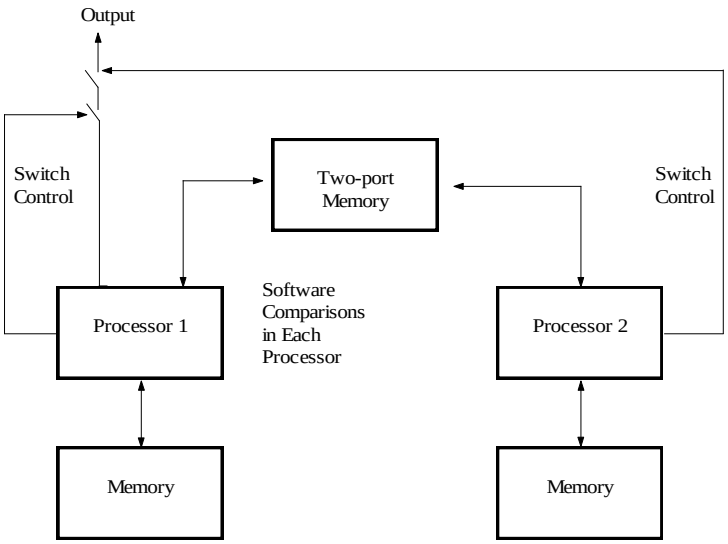


Figura 5.10

Os processadores realizam funções idênticas.

Se a memória compartilhada falha, ambos os processadores detectam a discrepância de valores com os valores de sua própria memória.
Se um processador falhar, a falha é detectada pelo outro processador pela discrepância de valores.
As chaves podem ser implementadas na forma “digital” ou “analogicamente” dependendo da aplicação.
Ambos os processadores devem concordar antes que o sistema permita produzir uma saída.

b) Unidade “Standby”.

A Configuração está apresentada na figura a seguir.

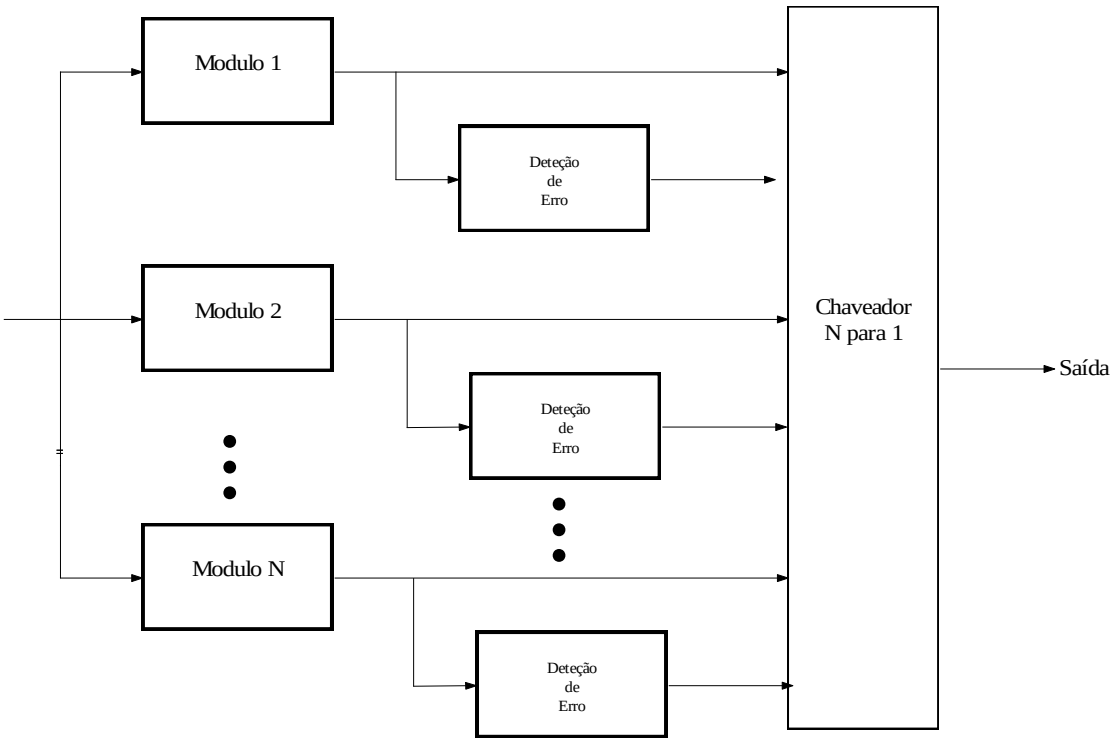


Figura 5.11

Nesta técnica um módulo é operacional e um ou mais módulos servem como reservas.
A reconfiguração neste caso pode ser vista conceitualmente como uma chave cuja saída é selecionada de um dos módulos bons. Se todos estão bons, pode-se fixar um esquema de prioridades.
Qualquer módulo com erro é eliminado desta consideração.

O sistema precisa tolerar uma interrupção durante o processo de reconfiguração.

Em função do tempo necessário pode-se ter “hot-standby” (opera em sincronia Ex: Controle de reação química) ou “cold standby”(não alimenta - Ex: satélite) reserva.

No caso “Hot Stand By” utiliza-se de unidades sobressalentes energizadas.

No caso “Cold Standy By” utiliza-se de unidades sobressalentes não energizadas.

Uma “questão chave” nesta abordagem é o esquema de Detecção de Erro/Detecção de Defeito usado para identificar o módulo em falha.

Duas técnicas são apresentadas a seguir:

b.1) “Pair-and-a-Spare”

A arquitetura está apresentada a seguir:

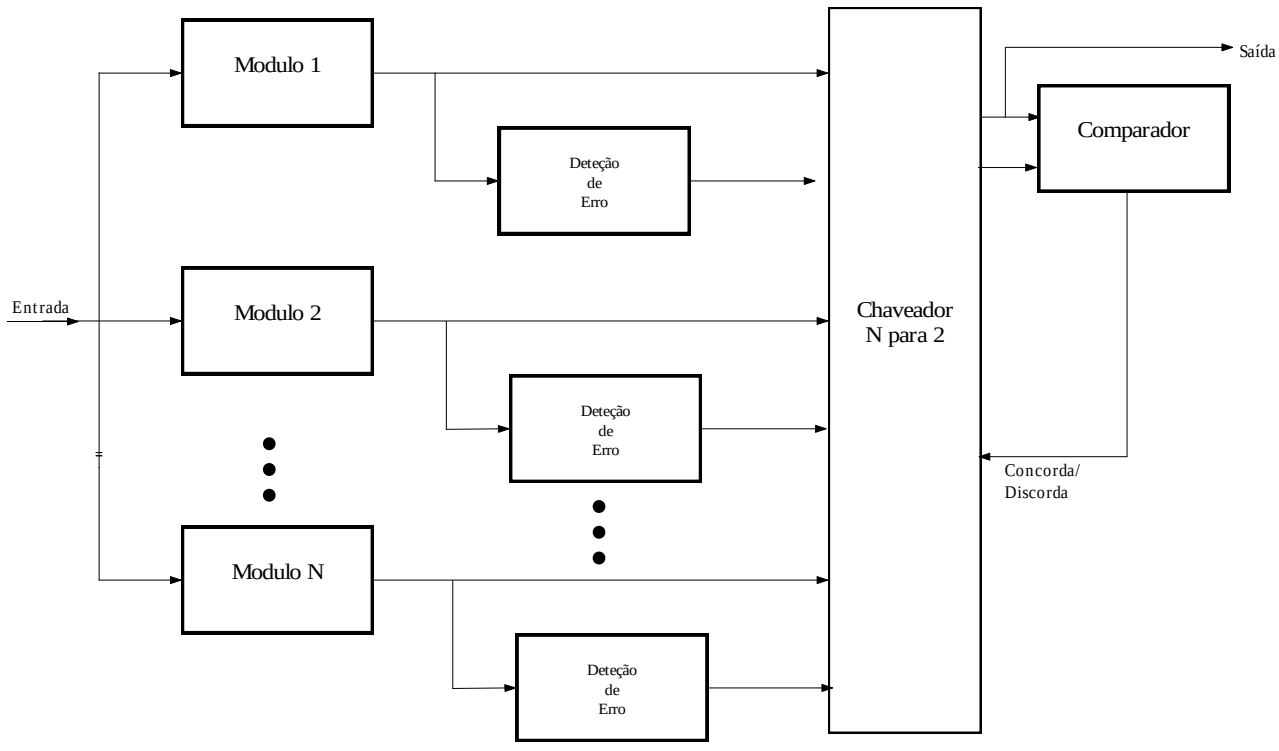


Figura 5.12

Nesta técnica dois módulos estão operando em paralelo durante todo o tempo e seus resultados são comparados para detectar erro. O sinal de erro da

comparação é utilizado para iniciar o processo de reconfiguração que vai remover o módulo em falha e o substituir á pelas reservas.
A chave usa a informação de erro do comparador e do módulo individual para manter dois módulos livres de erro operando em duplicação.
Outra saída seria usar os "módulos em par" e quando houver erro descarta o "par de módulos", em função da saída da comparação.

b.2) Temporizador " watchdog" (comparação com carteiro)

Neste caso é necessária uma ação para indicar o estado "livre de defeito".
A ausência da ação indica o defeito/erro.
O "watchdog" é um temporizador que deve ser resetado numa base repetitiva.
A falha do sistema em "resetar" resultará no sistema ser "resetado" ou "desligado" antes que outra falha aconteça no sistema.
A hipótese fundamental aqui é que o sistema seja livre de defeito se ele tem a capacidade de repetir periodicamente uma função como "resetar" um temporizador.
A freqüência que o temporizador deve ser "resetado" depende do sistema.
Exemplo:

- Sistema de controle de Avião < 100 ms;
- Sistema bancário < 1seg;
- Controle de metrô < 2 seg.

5.1.3. Redundância de Hardware Hibrida.

Combina as características da Passiva e Ativa. Usa-se Mascaramento de Defeitos e Reconfiguração (Detecção, Localização e Recuperação do Defeito).
É mais utilizada quando se exige um alto grau de integridade computacional. (Confiabilidade).

a) Redundância N-Modular Reservas.

N módulos arrumados numa configuração com votação. Além disso, sobressalentes são disponíveis para substituir as unidades em falha.
Esta técnica está apresentada na figura a seguir:

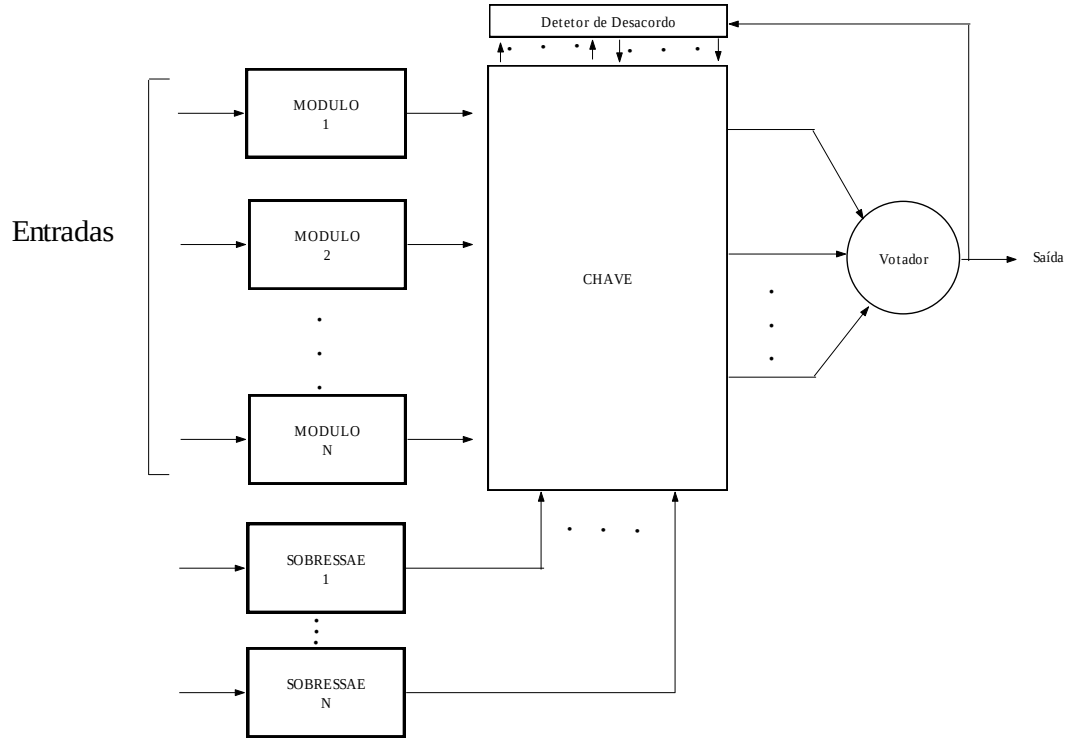


Figura 5.13

A configuração inicial permanece até que o detector de desacordo determine que exista uma unidade falha. Uma abordagem é comparar a saída do votador com as saídas dos módulos individualmente.

Neste caso a Unidade Reserva é recolocada no lugar do módulo falho.

b) Redundância com Auto-Remoção (“Self-Purging”)

A diferença é que neste caso todas as unidades participam ativamente, conforme ilustra a figura a seguir:

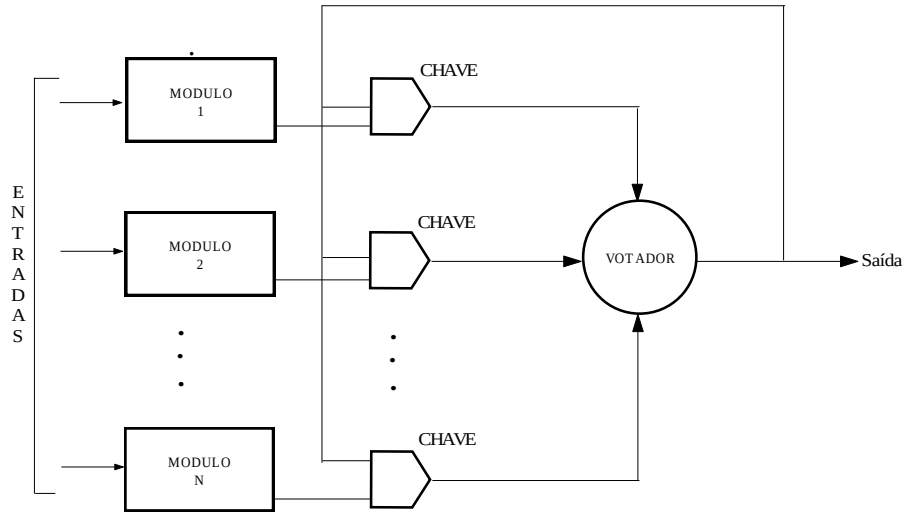


Figura 5.14

Cada módulo tem a capacidade de se auto remover do sistema no caso de sua saída discordar da saída do votador.

Neste caso o votador usa a técnica “threshold gate”. (através de componentes analógicos, na maioria).

xi... entradas binárias n...n° de entradas

z... saída binária

wi.. pesos

T... valor de “threshold” especificado.

$$Z = \begin{cases} 1, \text{ se } \sum_{i=1}^{n} wi \cdot xi \geq T \\ 0, \text{ se } \sum_{i=1}^{n} wi \cdot xi < T \end{cases}$$

Exemplos "Threshold gate" de 3 entradas.

wi = 1 e T = 2

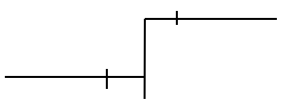
Entradas

X1	X2	X3	Σ	Saída
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	2	1
1	0	0	1	0
1	0	1	2	1
1	1	0	2	1
1	1	1	3	1

Uma abordagem é forçar para zero os pesos de todos os módulos em falha, não contribuindo neste caso para o valor de saída. A chave pode controlar o valor deste peso, associando o valor "zero" se a saída do módulo discorda da saída do sistema.

Implementação da chave utilizando Flip-Flop tipo JK:

J	K	Q (t + 1) *	Estado t+1
0	0	Q (t)	mantém
0	1	0	reset
1	0	1	set
1	1	Q (t + 1)	muda



A figura a seguir apresenta o esquema da chave:

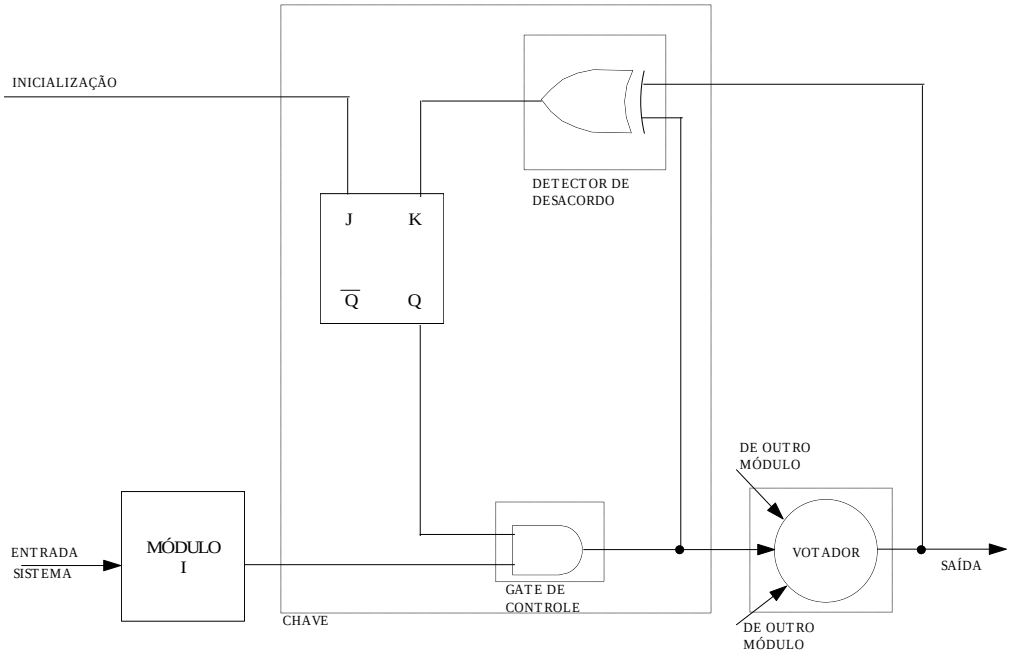


Figura 5.15

O mecanismo de iniciação permite um módulo, desabilitado uma vez, contribuir novamente no processo de votação, se ele for substituído por um sem falha.

- Exemplo: “Somador de 1bit”

Três somadores são usados para redundância tripla. Neste caso são necessários dois votadores: votador do bit soma e votador do bit carry. (vai um).

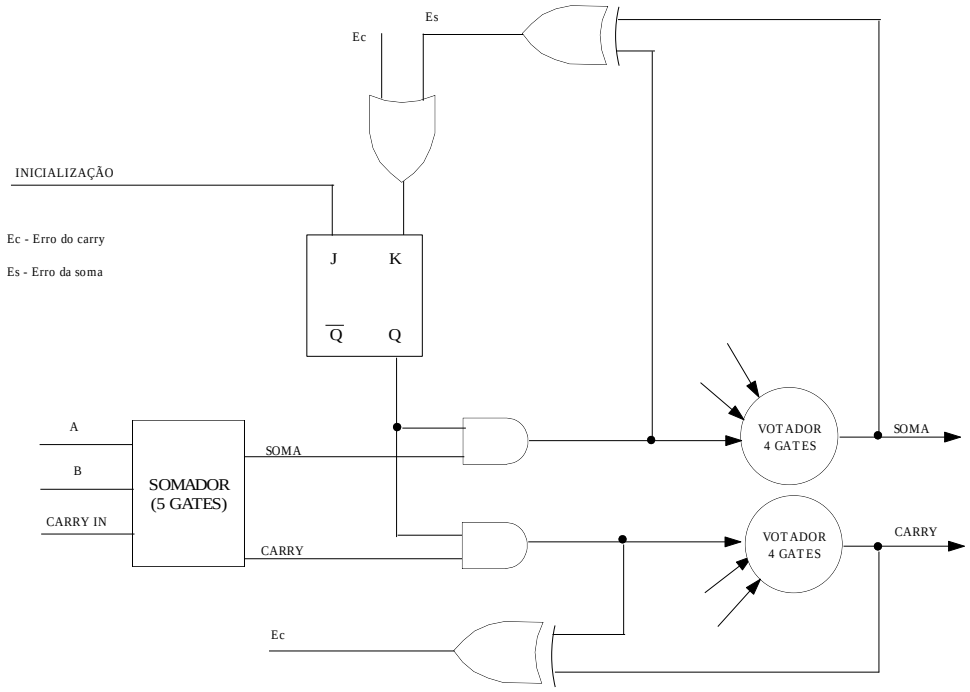


Figura 5.16

c) Redundância Modular "Sift-Out"

A estrutura básica está apresentada a seguir composta por comparadores, detectores e coletores.

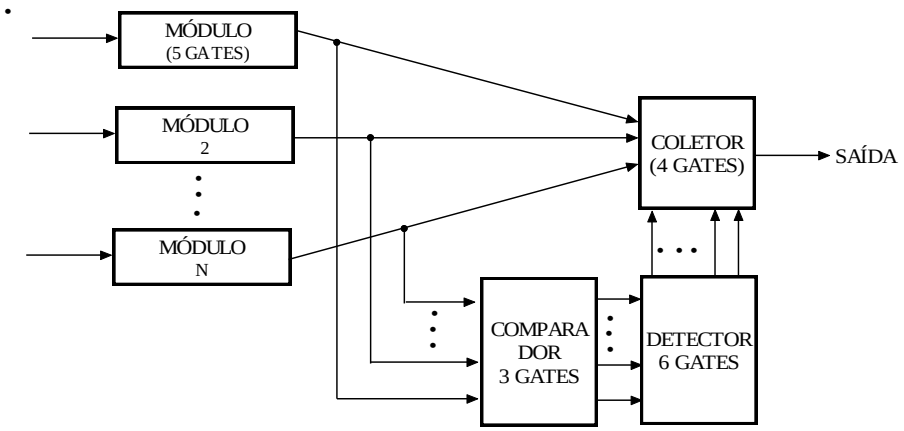


Figura 5.17

- Comparador: compara a saída de cada módulo com os demais. Ë produzido um sinal para cada comparação.

Cn, 2 = - - n ! - - comparações para n módulos.

2 ! (n-2) !

A saída é "1" se as duas saídas discordam;

A saída é “0” se as duas saídas concordam.

Exemplo para 3 módulos:

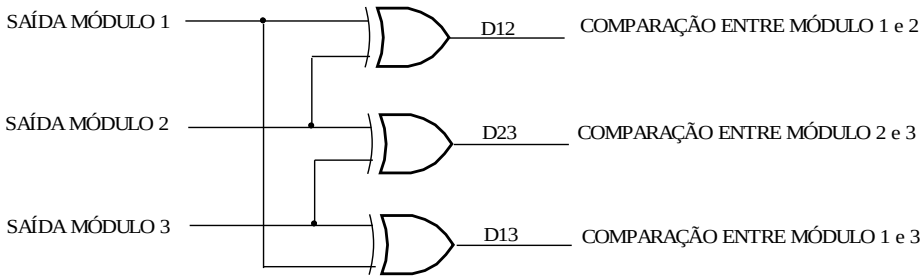


Figura 5.18

- Detetor: determina quais comparações discordam e desabilita a unidade (módulo) que discorda com a maioria dos módulos. A saída é para cada um dos módulos:

- “1”: o módulo discorda com a maioria dos módulos
- “0” o módulo concorda com a maioria dos módulos.

Usa o mesmo mecanismo de flip-flops tipo JK. Exemplo para 3 módulos está na figura 5.19.

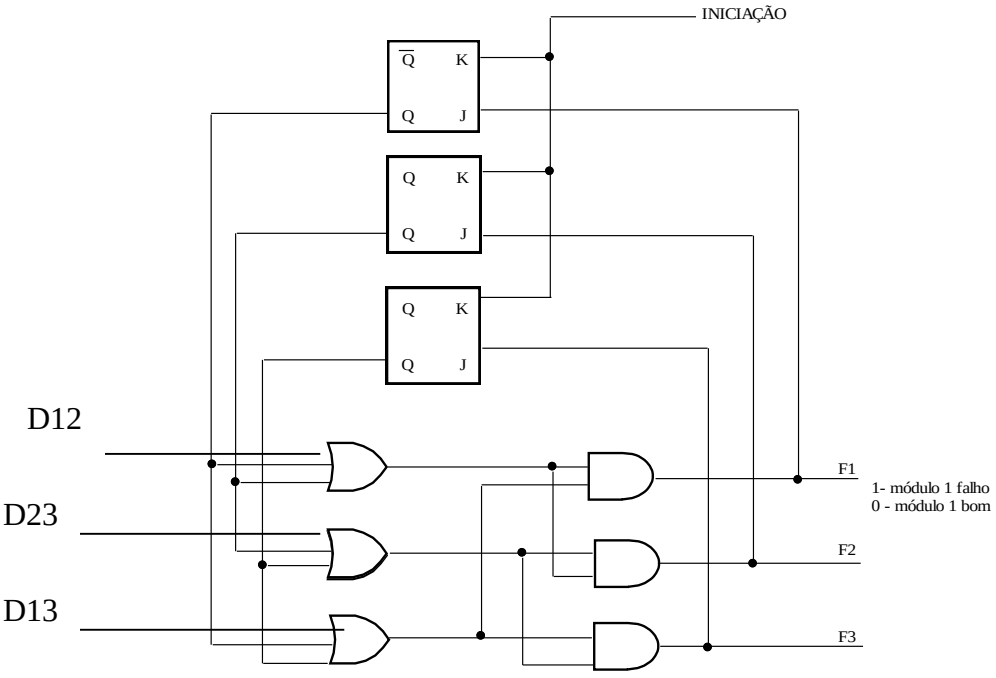


Figura 5.19 - Exemplo: Somador de 1 bit

O módulo que é indicado como falho não influencia na saída do sistema.

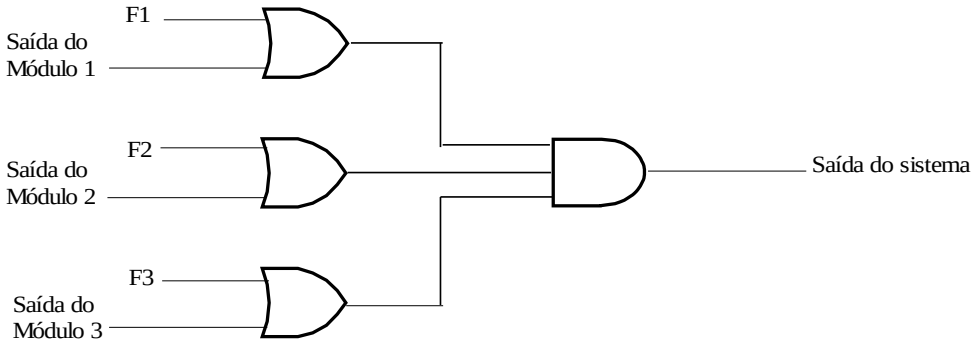


Figura 5.20 – Coletor

O esquema do coletor é apresentado na figura 5.20.

O sistema como um todo é apresentado na figura 5.21.

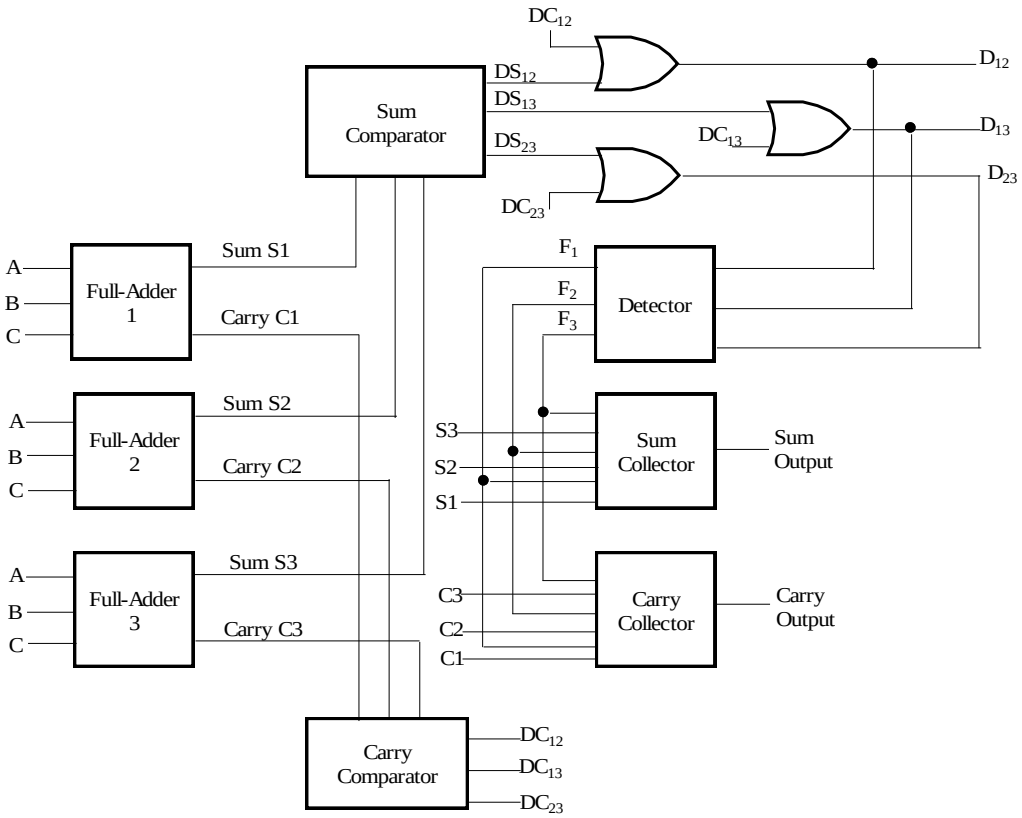


Figura 5.21

COMPARAÇÃO	"AUTO REMOÇÃO"	SIF-OUT"
No de gates: no. de flip flops:	38 3	38 * ← interessante 3

d) Arquitetura Tripla-Duplex.

Combina redundância triplicada com comparação.

O TMR permite mascarar defeitos. A duplicação com comparação permite defeitos serem detectados e módulos com defeitos removidos da votação.

O esquema desta solução é apresentado a seguir:

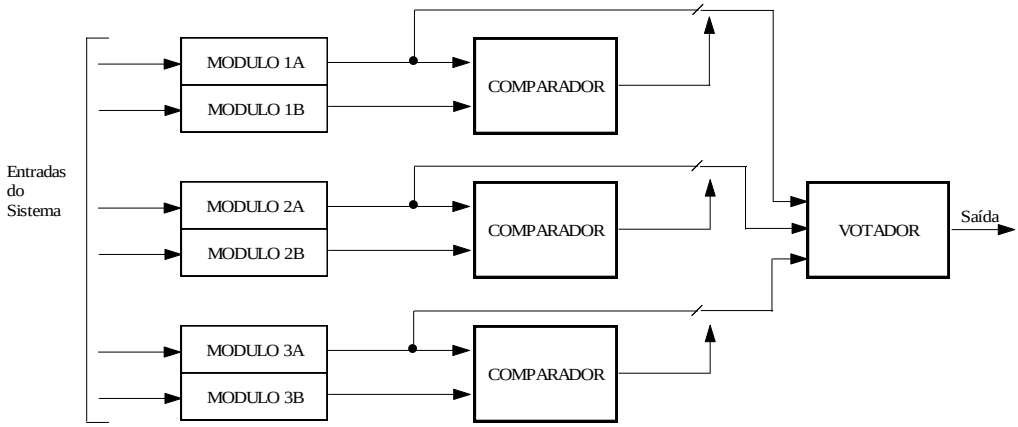


Figura 5.22

5.2. Redundância de Informação

É adição de informação para permitir detecção de defeito, mascaramento de defeito ou possivelmente tolerância ao defeito.

EX.: código de detecção e correção de erros.

Algumas definições:

- Código: meio de representar informação, ou dados, usando um conjunto bem definido de regras.

- Palavras de Código: coleção de símbolos usados para representar um pedaço particular de dados baseados num código específico.

- Código Binário: os símbolos são "0" s e "1" s.

Um conceito fundamental na caracterização dos códigos de detecção e correção de erros é a "Distância Hamming".

A "Distância Hamming" entre duas palavras binárias é o número de posições de bits na qual as duas palavras diferem.

0000 x 0101 dist. hamming = 2.

Distância de código.

É a mínima distância hamming entre duas palavras de código válidas.

Exemplo:

Distância de código = 3

Qualquer erro em único bit pode ser corrigido, pois vai ter uma distância de hamming de 1 do código correto e uma distância de hamming de pelo menos 2 dos demais códigos válidos.

Regra geral:

c... No de bits que podem ser corrigidos de erros.

d-.. No de bits que podem ser detectados erros.

Hd- distância de Hamming

$2c + d+1 \leq Hd$

Código Separável: para obter o código original basta retirar os bits de código ou de verificação.

Código não Separável: para obter o código original é necessário um processamento mais complexo.

5.2.1. Código de Paridade

Código de paridade de um bit requer a adição de apenas um bit á palavra.

Se o bit extra resulta num número total de "1" s ser impar, o código é referenciado como paridade impar.

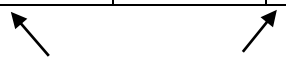
Se o resultado do No de "1" s é par, o código é de paridade par.

Ambos tem uma distância de Hamming de 2 permitindo a detecção de erro em 1 bits, mas não a correção.

Trata-se de um código separável.

Exemplo:

Digito Decimal	BCD	BCD c/ Paridade	Impar	BCD c/ Paridade	Par
0	0000	0000	1	0000	0
1	0001	0001	0	0001	1
2	0010	0010	0	0010	1
3	0011	0011	1	0011	0
4	0100	0100	0	0100	1
5	0101	0101	1	0101	0
6	0110	0110	1	0110	0
7	0111	0111	0	0111	1
8	1000	1000	0	1000	1
9	1001	1001	1	1001	0



bit de paridade

Aplicação: Memórias em sistemas computacionais.

O esquema de aplicação é mostrado a seguir:

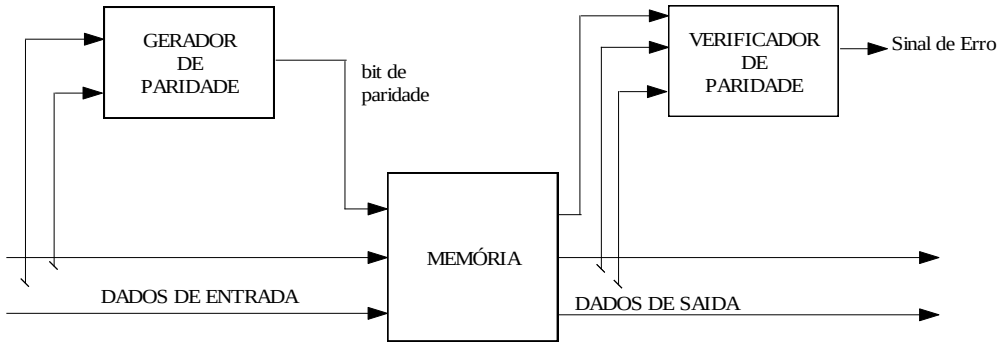
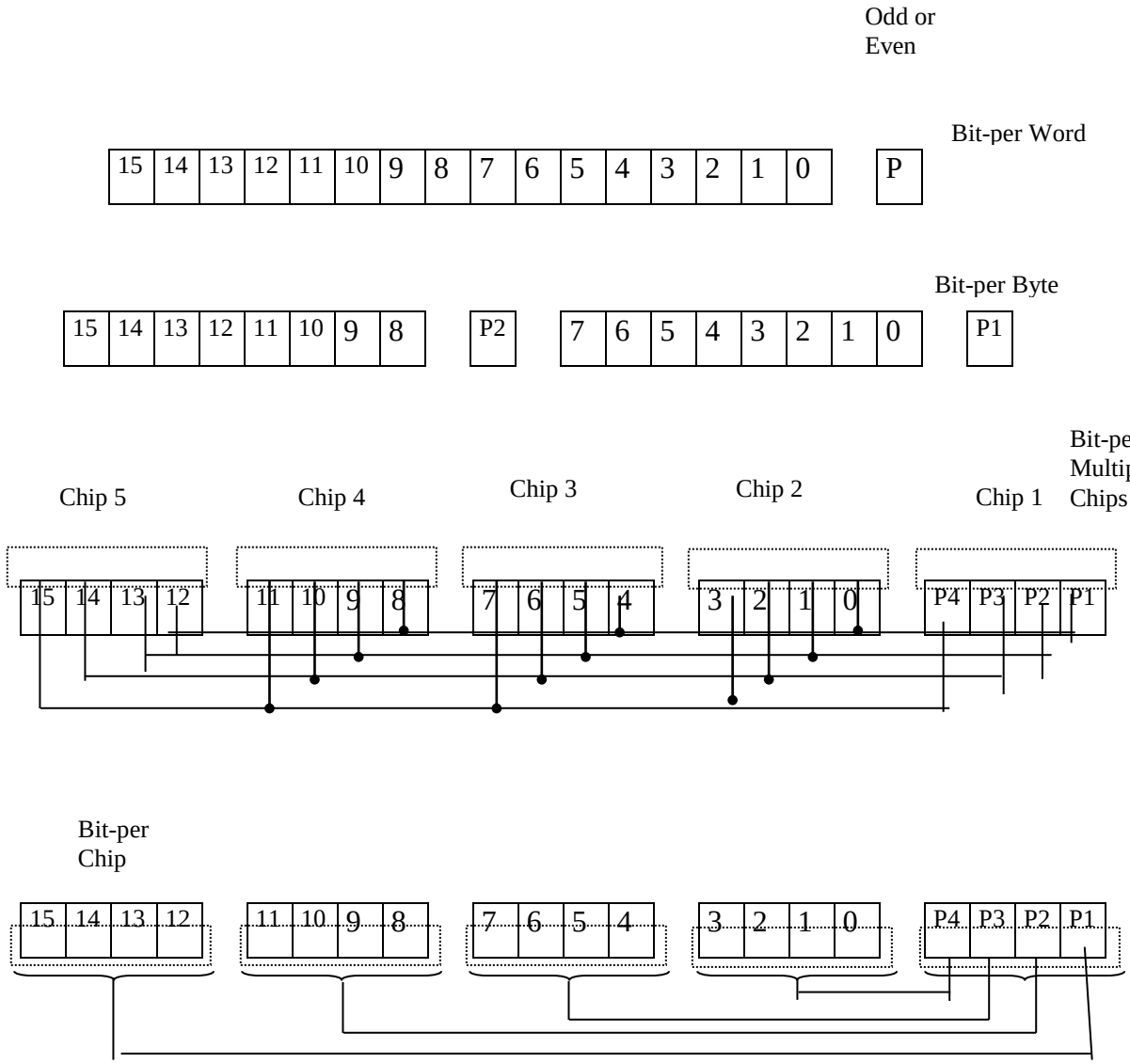


Figura 5.23

Existem outros tipos de paridade para detectar mais erros no código:

5. Paridade bit-por-palavra;
5. Paridade bit-por-byte;
5. Paridade bit-por-chips;
5. Paridade bit-por-multiplos-chips;
5. Paridade interlaçada.

A figura a seguir apresenta o esquema geral destes bits de paridade


Odd or
Even
Bit-per Word
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
P
Bit-per Byte
15 14 13 12 11 10 9 8
P2
7 6 5 4 3 2 1 0
P1
Bit-per
Multiple-
Chips
Chip 5
Chip 4
Chip 3
Chip 2
Chip 1
15 14 13 12
11 10 9 8
7 6 5 4
3 2 1 0
P4 P3 P2 P1
Bit-per
Chip
15 14 13 12
11 10 9 8
7 6 5 4
3 2 1 0
P4 P3 P2 P1

Interlaced
Parity

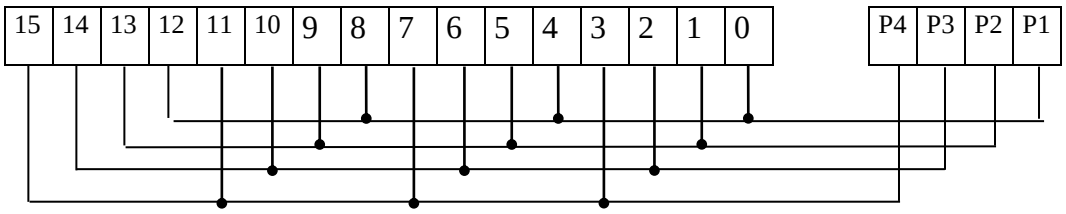


Figura 5.24

a) Bit por palavra

Quando o bus falha em "1", inclusive o bit de paridade, a paridade Impar não detecta.

Quando o bus falha em "0", inclusive o bit de paridade, a paridade par não detecta.

b) Bit por Byte -1 grupo paridade impar.

-1 grupo paridade par.

Falha tudo 1 → paridade par detecta

Falha tudo 0 → paridade impar detecta.

Ambos, "bits por palavra" e "bit por Byte", não detectam erros múltiplos, no caso de falha de uma pastilha de memória contendo bits de um dado, pois a distância de código é um.

Para resolver este fato tem-se:

c) Bit por múltiplos Chips

Detecta erro numa pastilha, total da memória todos os bits de paridade detectam este erro (podem detectar) usando tipos de paridade diferentes.

Pode ser difícil de detectar os Chips com defeito.

Surgiu então,

d) Bit por chip tem mesmo problema de bit por palavra.

e posteriormente surgiu,

e) Bit com Paridade Interlaçada ("overlapping").

Os grupos não estão relacionados com as pastilhas físicas. Muito usado quando erros em bits adjacentes são mais prováveis. Ex.: "bus" paralelo.

A paridade interlaçada apresenta grupos que são formados com cada bit aparecendo em mais de um grupo.

Neste caso o erro pode ser localizado além de ser detectado.

Este tipo de paridade é o conceito do código de correção de erro Hamming.

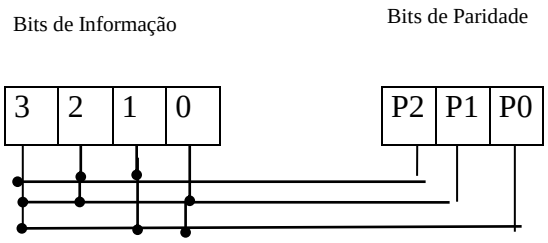
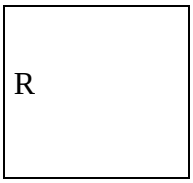


Figura 5.25

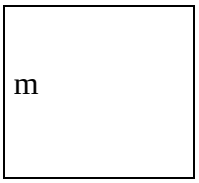
Erro no bit		Erro no bit de paridade				
3	→	P2	P1	P0	111	7
2	→	P2	P1		110	6
1	→	P2		P0	101	5
0	→		P1	P0	011	3
P2	→	P2			100	4
P1	→		P1		010	2
P0	→			P0	001	1
sem erro	→				000	0

4 bits ⟶ 3 bits de paridade.

Pensamento;



R bits
informação



m bits de paridade

n°de erros: R+m
sem erro : 1

$R+m+1 \leq 2^m$

n° de bits de Paridade/ n° de bits de informação

n° de bits de Informação	n° de bits de Paridade	Porcentagem de Redundância (%)
2	3	150
4	3	75
6	4	66,7
8	4	50,0
10	4	40,0
12	5	41,7
16	5	31,25
24	5	20,8
32	6	18,75
64	7	10,9

5.2.2. Códigos m de n

Definir palavras que tem n bits com exatamente m 1's.

→ altera

∀ erro → n° DE 1'S → M+1 OU M-1

Em um único bit.

Dificuldades: processo de codificação, decodificação e detecção de erro.

Exemplo: 3 de 6 (i de 2 i) (Redundância neste caso → 100%)

Informação	Código 3 de 6	
000	000	111
001	001	110
010	010	101
011	011	100
100	100	011
101	101	010
110	110	001
111	111	000

Informação adicional

Vantagem Código Separável

Distância é 2

Este código detecta erros simples em bits e erros múltiplos unidirecionais (1 0; ou 0 1).

Exemplo: 2 de 5 para dados BCD (não separável)

	BCD	**2 de 5**	
0	0000	00011	Qual a regra geral?
1	0001	11000	
2	0010	10100	
3	0011	01100	
4	0100	10010	
5	0101	01010	
6	0110	00110	
7	0111	10001	
8	1000	01001	
9	1001	00101	
	↓	↓	
	d3 d2 d1,d0	b4 b3 b1 b0	

Exemplo:

do = b4.b3 + b3.b2 + b3.b1 + b4.bϕ + b2.bϕ

= b3 (b4+b2) + bϕ (b4+b2) + b3.b1

= **(b4+b2).(b3+bϕ)+b3.b1**

d1 = b4.(b2+bϕ) + b2.(b3+b1)

5.2.3. Códigos Duplicados

Duplica o código original para formar a palavra codificada.

. Vantagem: simplicidade

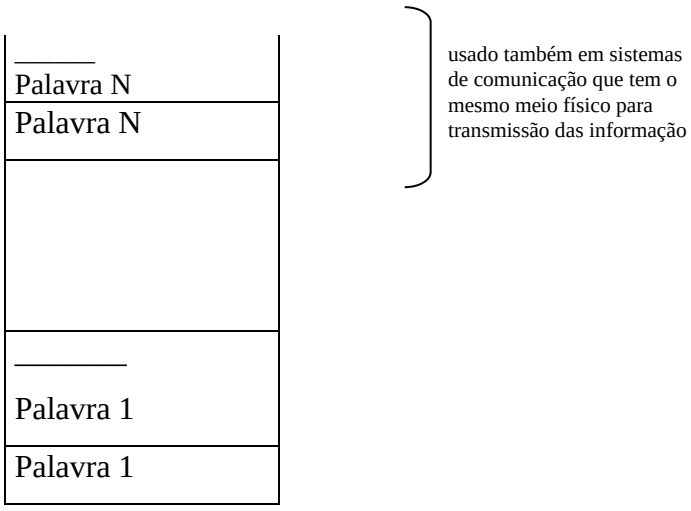
. Desvantagem: n° de bits extra

Aplicação: sistema de memória e alguns sistemas de comunicação.

Em sistema de comunicação o conceito de duplicação é freqüentemente aplicado na transmissão de toda a informação duas vezes. Se as cópias concordam, a informação é assumida estar correta.

O prejuízo é o decréscimo na taxa de informação.

Outra variação é complementar a porção duplicada da palavra codificada.



Outra variação é a duplicação (inverte e compara) “swap and compare”. Neste método são mantidas as duas cópias da informação original, mas trocam-se as metades da palavra codificada.

• • •	• • •
L2	U2
U2	L2
L1	U1
U1	L1

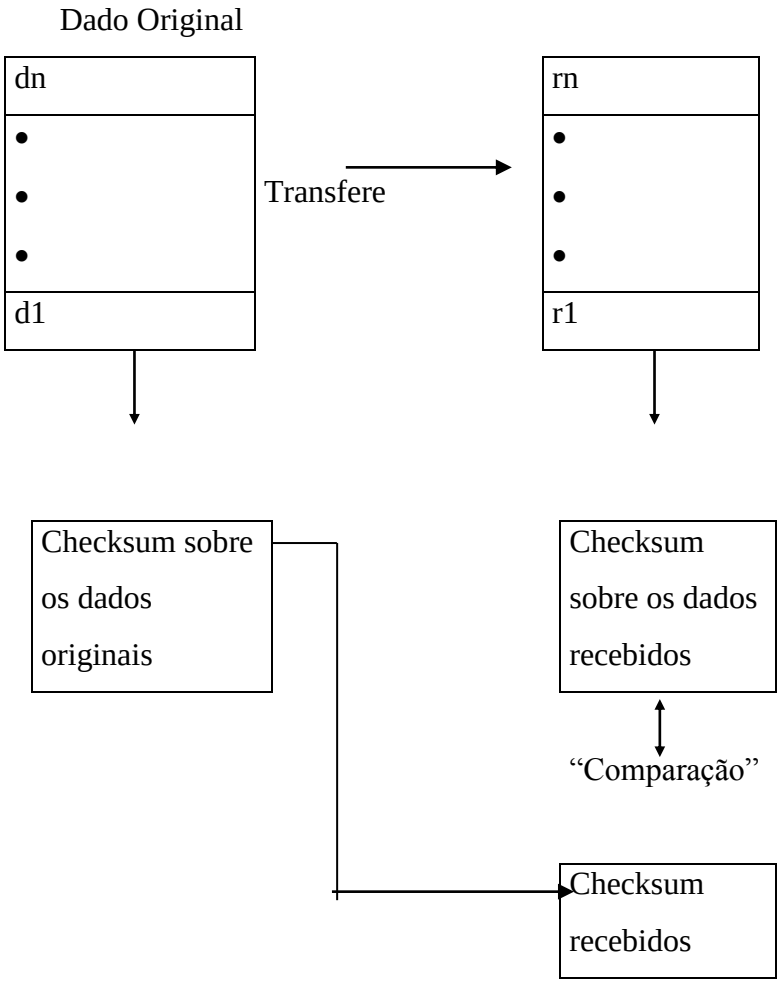
Ui - Metade Superior

Li - Metade Inferior

5.2.4. Checksums (Código Separável)

Aplicável quando blocos de dados devem ser transmitidos de um ponto para outro.

Conceito básico: Soma dos dados originais (variações na forma que a soma é feita).



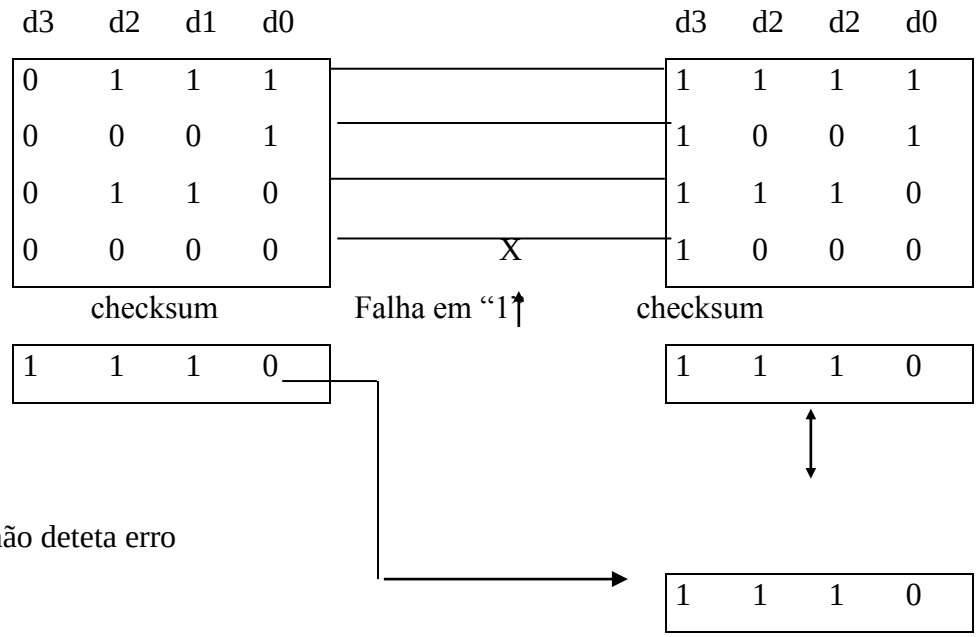
Há quatro tipos primários de checksum:

- Precisão simples;
- Precisão dupla;
- Honeywell;
- Residual.

a) Precisão Simples

Ignora o overflow do checksum.

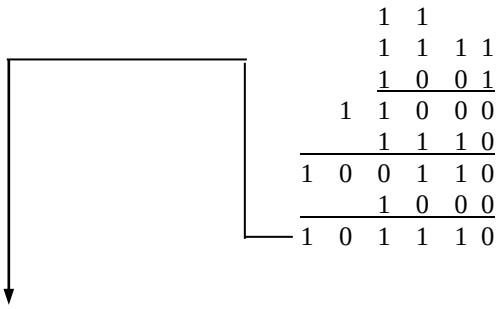
A primeira dificuldade é que se perde a habilidade de detectar erros



b) Precisão Dupla

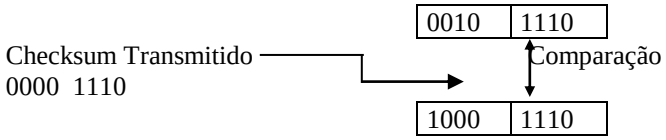
Computar um checksum de 2n bits para um bloco de palavras com n bits.

Ainda é possível overflow, mas, mais difícil.



Exemplo anterior

Checksum Calculado

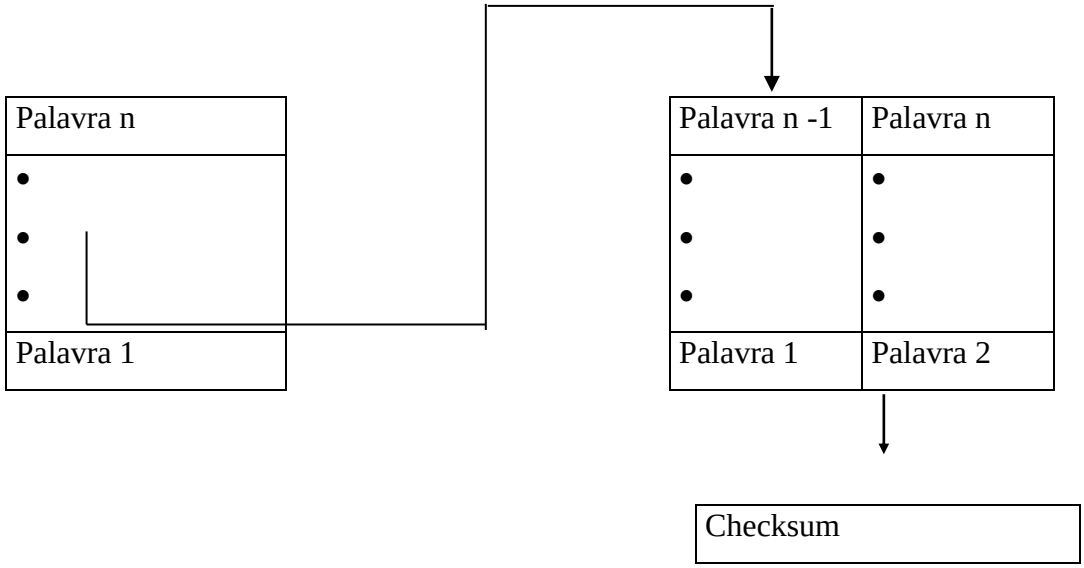


c) Honeywell

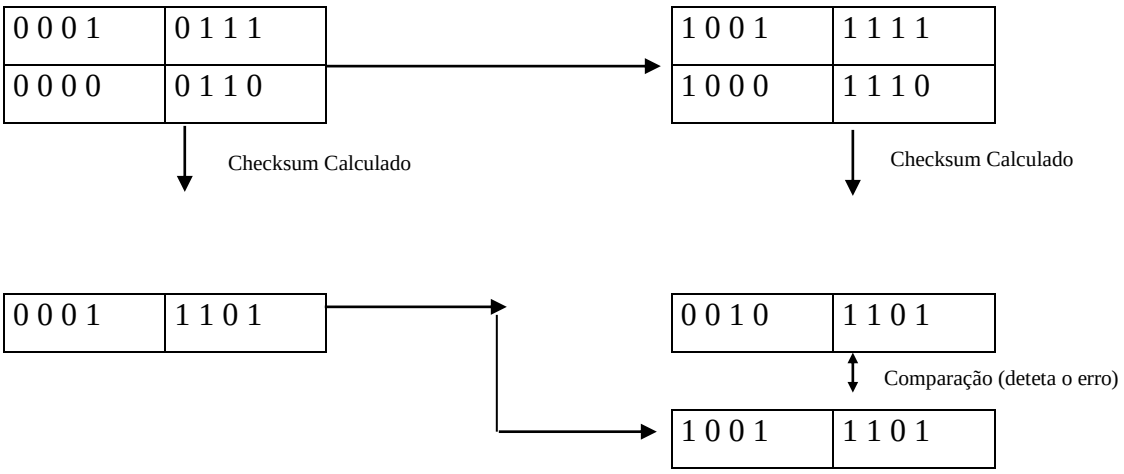
Concatena palavras consecutivas para formar uma coleção de palavras de comprimento duplo.

O Checksum é calculado sobre esta nova estrutura.

Vantagem: erro num bit de todas as palavras provocará erro pelo menos em 2 bits do Checksum.



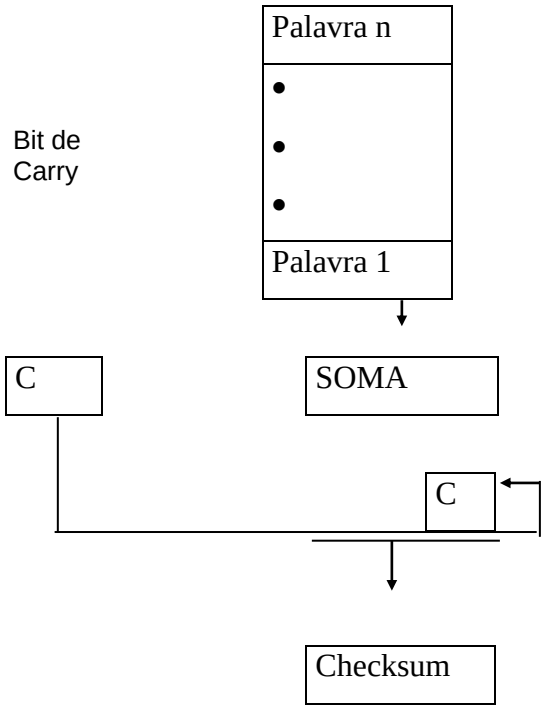
Exemplo anterior:

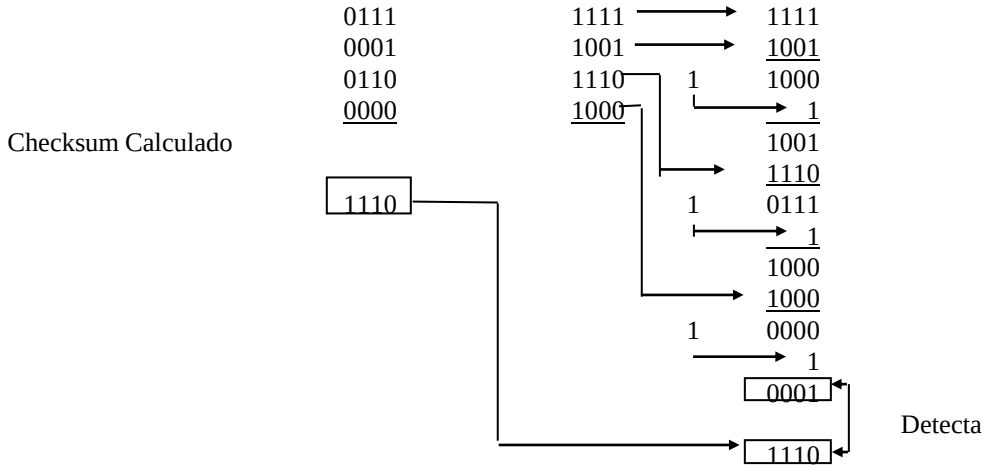


d) Residual

Mesmo conceito da precisão simples exceto que o bit carry não é ignorado, mas, é adicionado de volta ao checksum.

Exemplo:





Através do checksum não há condições para se determinar onde o erro ocorreu.

5.2.5. Códigos Cíclicos (Não Separável).

É caracterizado pelo seu polinômio gerador G (x) de grau maior ou igual a (n-R) onde n é o número de bits contidos na palavra codificada produzida por G (x), e R é o número de bits na informação original a ser codificada.

Tais códigos têm a propriedade de serem capaz de detectar todos os erros simples e múltiplos, adjacentes, que afetam menos do que (n-R) bits.

A propriedade de detecção de erro dos códigos cíclicos é particularmente importante nas aplicações de comunicações onde erros transitórios podem ocorrer.

No código cíclico a codificação representa os coeficientes do polinômio.

Exemplo: Código = $v_{n,.....}v1$, v0 corresponde ao polinômio

$V(x) = Vo+v1.X+v2.X^2+...+vn-1X^n$. (Polinômio Codificado)

V (X) = D (X).G(X). Polinômio de dados é D (x).

Qualquer adição requerida durante a multiplicação de dois polinômios é desempenhada usando módulo-2.

Exemplo: $G(X) = 1+X+X^3$

$D(X) = 1+X+X^2+X^3$

v (x) = ? 1011

1111 $v(x) = 1+X^3+X^5+X^6$

1011

1011 1101001

1011 ↓

1011 forma de representar

1101001

Information (d_0,d_1,d_2,d_3)	Code (v_0, v_1, v_2, v_3, v_4, v_5, v_6)
0000	0000000
0001	0001101
0010	0011010
0011	0010111
0100	0110100
0101	0111001
0110	0101110
0111	0100011
1000	1101000
1001	1100101
1010	1110010
1011	1111111
1100	1011100
1101	1010001
1110	1000110
1111	1001011

Distância de código = 3

↓

* Qualquer erro em 2 bits é detetado *

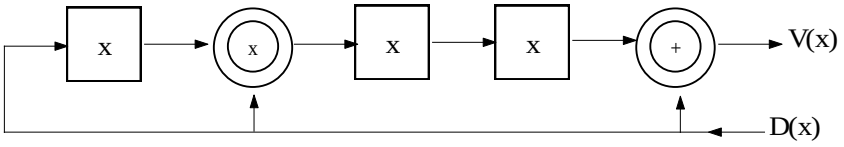
O circulo lógico é obtido da seguinte forma:

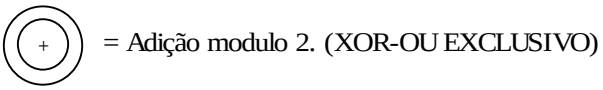
Ex.: $G(X) = 1+ X^2+ X^3$; $D(X)$

$V(X) = G(X).D(X)=(1+ X^2+ X^3)\ D(X) =$

$(D (X)+D(X)\ X^2 +D(X).\ X^3 =$

$(D(X)+(D(X)+D(X).X).\ X^2=$





Exemplo:

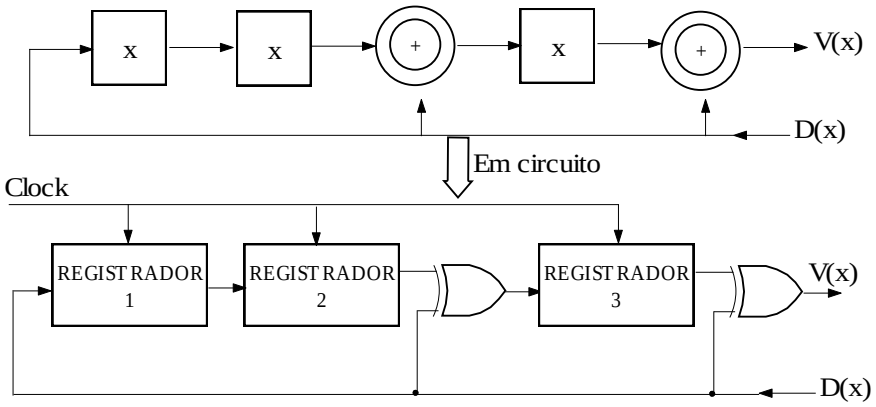
$G(x) = 1 + X + X^3$

$D(x)$

$V(x) = D(x) + (1+X+ X^3)$

$= D(x) + (D(x) . X + D(x) . X^3$

$= D(x) + (D(x) + D(x).\ x^2) . X$



Processo de Codificação Registradores: D(X)= 1011

Registradores

Clock	1	2	3	D(X)	V(X)
0	0	0	0	1	1
1	1	0	1	1	0
2	1	1	1	0	1
3	0	1	1	1	0
4	1	0	0	0	0
5	0	1	0	0	0
6	0	0	1	0	1
7	0	0	0	0	0

↑

1011

1011

1011

1011

1011

1000101

Processo de Decodificação

Se R(X) é um código válido recebido então:

R(X) = D(X) . G(X)

Podemos escrever que

⟶ Síndrome

R(X) = D(X). G(X) + S(X)

e que S(X) deve ser zero se o polinômio R(X) é um código válido.

V(X) = G (X).D(X)

Exemplo: $G(X) = 1+X+X^3$

$D(X) = V(X)+B(X)$

$V(X) = (1+X+X^3).D(X)+S(X)$

$V(X) = D(X)+D(X)(X+X^3)$

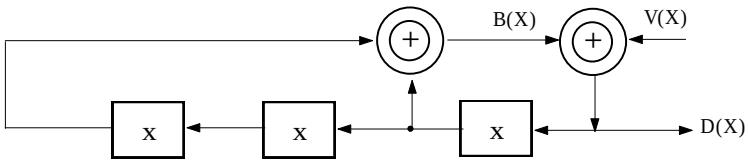
Em módulo-2 Soma e Subtração são iguais

- $D(X) = V(X) + D(X)(X+X^3)$

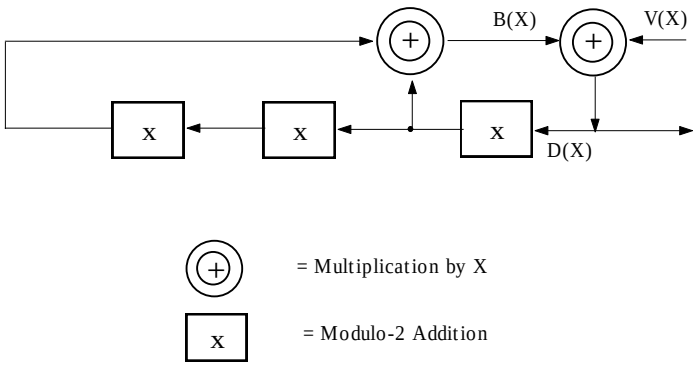
$B(x) = D(X)(X+X^3)$

$= D(X) . X+D(X) . X^3$

Circuito:



Clock period	Register values					
	1	2	3	V (x)	B(x)	D(x)
0	0	0	0			
				1	0	1
1	0	0	1			
				0	1	1
2	0	1	1			
				1	1	0
3	1	1	0			
				0	1	1
4	1	0	1			
				0	0	0
5	0	1	0			
				0	0	0
6	1	0	0			
				1	1	0
7	0	0	0			



5.2.6. Códigos Aritméticos.

São úteis para verificar operações aritméticas (+,*,).

Propriedade: A(b*c) =A(b)* A (c).

Exemplos de códigos aritméticos: AN, Residual, Inverso - residual e Sistema de Números Residuais.

a) Códigos AN

O mais simples código aritmético que é formado multiplicando cada dado por uma constante A.

Soma: A 2^a (Não pode ser potência de 2).

No lado da recepção verifica se o código é divisível por A.

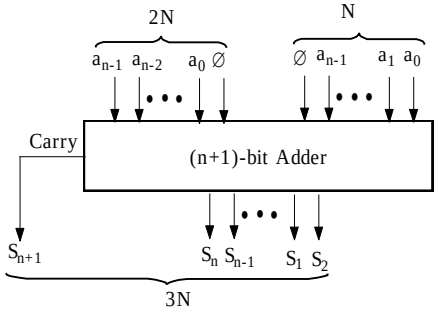
(a_{n-1} a_{n-2}.....a_2 a_1 a_0).

$a_{n-1}. 2^{a+n-1}+ ...+ a_2. 2^{a+2} +a_1. 2^{a+1}+a_0. 2^a +0. 2^a +0. 2^{a-1} --+0. 2^0.$

mudando este bit continua sendo divisível por 2^a

Exemplo: 3N código = (N+2N)

Original Information	3/V code word
0000	000000
0001	000011
0010	000110
0011	001001
0100	001100
0101	001111
0110	010010
0111	010101
1000	011000
1001	011011
1010	011110
1011	100001
1100	100100
1101	100111
1110	101010
1111	101101



b) Códigos Residuais (Separável)

Adiciona um número residual ao número.

O número residual é simplesmente o resto quando o número é dividido por um inteiro.

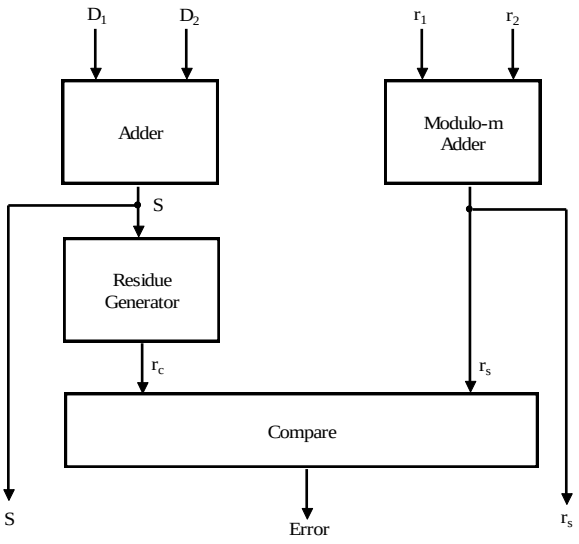
Ex.: Inteiro N, inteiro m

$$\begin{cases} N = Im + r \\ 0 \leq r < m. \end{cases}$$

base (módulo)

Information	Residue	code word	
0000	0	0000	00
0001	1	0001	01
0010	2	0010	10
0011	0	0011	00
0100	1	0100	01
0101	2	0101	10
0110	0	0110	00
0111	1	0111	01
1000	2	1000	10
1001	0	1001	00
1010	1	1010	01
1011	2	1011	10
1100	0	1100	00
1101	1	1101	01
1110	2	1110	10
1111	0	1111	00

Processo de detecção de erro na recepção.



5.2.7. Códigos Berger

Adicionar um conjunto de bits verificadores, caracterizando um código separável.

O comprimento é _N e tem _I bits de informação e _R bits de verificação.

$R=\log_2 (I+1)$ e $n=I+R$

Exemplo:

(0111010) ⟵ Informação I=7

R=log (7+1)=3 ⟶ ⇩

O no. de "1"s é 4 ⟶ (100)

⇩ complemento

(011)

⇩ código resultante

(0111010011)

Quando o n de bits da Informação é baixo, a redundância é alta.

N de bits de Informação (I)	N de bits de verificação (R)	%
4	3	75
8	4	50,00
16	5	31,25
32	6	18,75
64	7	10,94

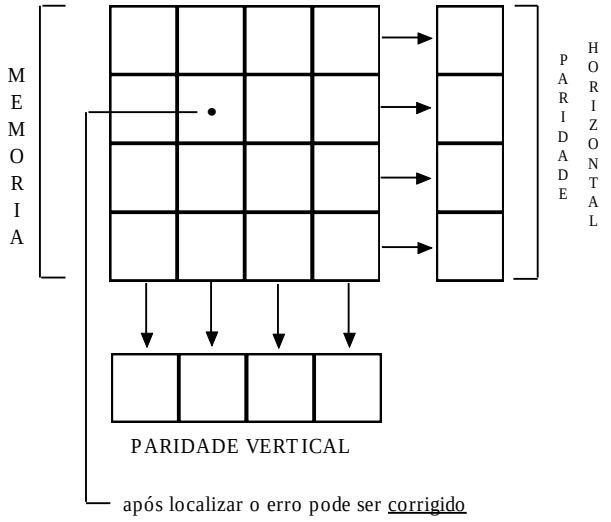
Original Information	Berger code	
0000	0000	111
0001	0001	110
0010	0010	110
0011	0011	101
0100	0100	110
0101	0101	101
0110	0110	101
0111	0111	100
1000	1000	110
1001	1001	101
1010	1010	101
1011	1011	100
1100	1100	101
1101	1101	100
1110	1110	100
1111	1111	011

Exemplo: I=4 e R=3

1011 → no. de "1" e = 3 → (011)

(1011100) ← (100)

5.2.8. Paridade Horizontal e Vertical



Quando o erro é múltiplo, é possível detectar, mas não corrigir o erro.

5.2.9. Código de Correção de Erro Hamming

Requer de 10 a 40% de redundância. A codificação e codificação acrescentam um atraso relativamente pequeno.

Este código utiliza _c bits de verificação de paridade para proteger R bits de informação.

Relação: $2^c \geq c + R + 1$

Comprimento total: n = c + R

O código Hamming é formado particionando os bits de informação em grupos de paridade e especificando um bit de paridade para cada grupo. (par ou impar).

A habilidade de localizar qual bit é errado é obtida através de superposição dos grupos de bits.

Exemplo:

bits de informação: (d_3 d_2 d_1 d_0)

$2^c \geq c + 4 + 1 = C+5$

$2^c \geq c + 5 \rightarrow C = 3$ (C_1 C_2 C_3)

Bit errado	bit de Verificação Afetado
d0	C1, C2
d1	C1, C3
d2	C2, C3
d3	C1, C2, C3
C1	C1
C2	C2
C3	C3

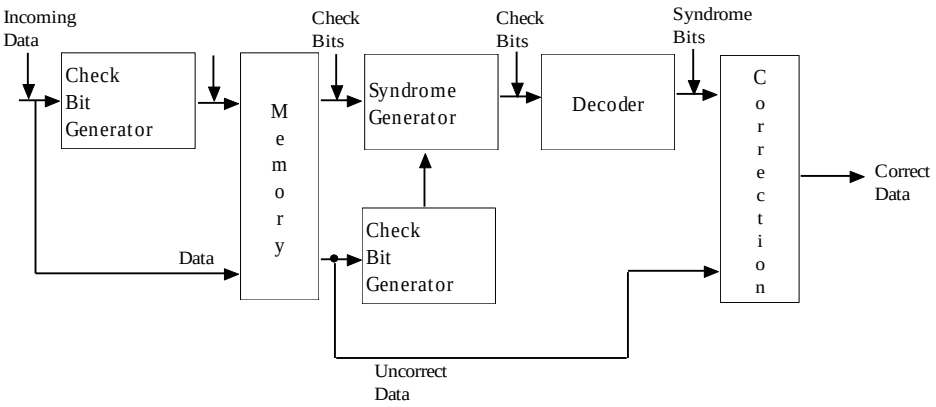
Na recepção calcula-se o código de verificação e compara com o recebido.

É definido SINDROME como a comparação entre o código calculado e o recebido (XOR).

Se for igual a "1" indica incorreção.

(Um Bit errado)	C1, C2, C1, C3
	Sindrome
d0	110
d1	101
d2	011
d3	111
c1	100
c2	010
c3	001

Esquema geral utilização



Para 2 bits errados acrescenta-se mais um bit de paridade da palavra. (Parid. Global).

Sindrome	ParidadeGlobal	correção
não zero	incorreto	um bit errado → correção
não zero	correto	Dois bits errados →
zero	correto	Correto

5.2.10 Circuitos Integrados com Correção de Erro

Muitos circuitos integrados estão disponíveis para detectar, localizar e corrigir erros. Como exemplo, pode-se citar o 8206 da Intel, MC68540 da Motorola, AM2960 e AMZ8160 da Advanced Micro Devices, DP8400 da National Semicondutor e MB1412A da Fijitsu.

Estes circuitos integrados são projetados para gerarem os bits de verificação, gerar a "sindrome", decodificar a "sindrome" e realizar a correção dos dados. Unidades típicas são organizadas para suportar dados de 16 bits, mas podem ser expansíveis para detecção e correção de palavras de tamanho maior.

Quase todos os CI' s usam o Código de Hamming modificado para detectar erros duplos e corrigir erros simples.

O Diagrama em blocos da estrutura fundamental de um Circuito Integrado deste tipo é mostrada a seguir. Uma palavra vindo do sistema vai diretamente á memória e a um gerador de bits de verificação. Ambos são armazenados na memória. Na leitura de uma palavra da memória ocorrem os seguintes eventos:

1. O dado é usado para regenerar os bits de verificação através de um Gerador de Bits de Verificação.
2. É criado a "síndrome" comparando os bits de verificação original com os regenerados.
3. A "Síndrome" é usada para identificar o bit errado.
4. Os dados são passados através de uma unidade de correção.
5. Os dados corrigidos são colocados no bus.

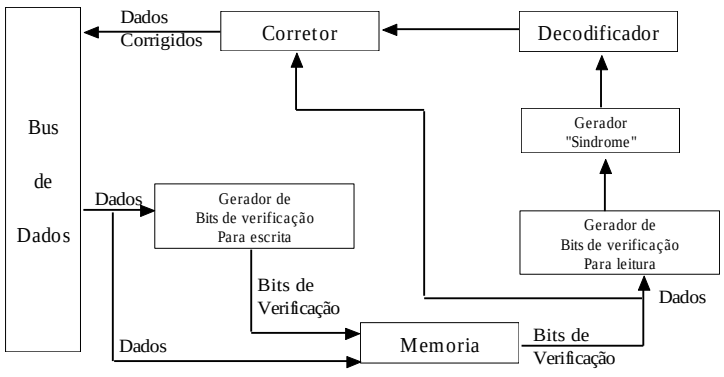
O número de ocorrência de erros é usado para indicar um defeito permanente na memória.

Para Intel 8206: (16 bits)

- tempo de detecção do erro: $\leq$ 52ns
- tempo de deteção e correção: $\leq$ 67ns.

Para efeito de rapidez, em códigos separáveis, a informação já pode ser usada enquanto se realizam as verificações necessárias.

Circuito Integrado de Correção de Erro



5.3. Redundância por Tempo.

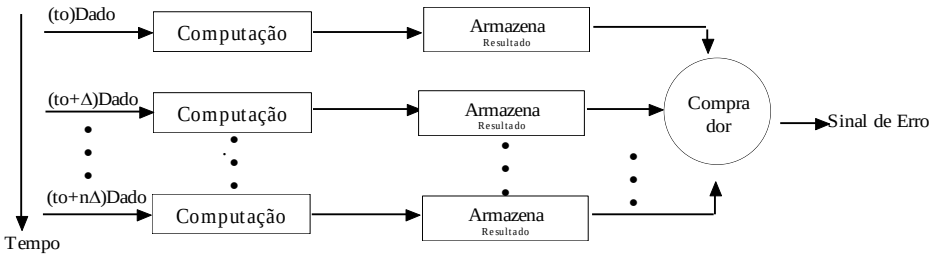
Tanto a redundância de hardware como a de informação, requer um hardware extra na sua implementação.

A Redundância por tempo, por sua vez, não requer hardware extra.

A escolha adequada deve ser tomada em função dos requisitos de cada aplicação.

5.3.1. Detecção de Defeito Transiente.

O conceito básico da redundância temporal é a repetição da computação de forma a permitir que o defeito seja detectado. A forma básica da redundância temporal é apresentada a seguir:



Se um erro é detectado, a computação pode ser realizada novamente para verificar se a discordância contínua ou desaparece. Tal abordagem é interessante para detecção de erros resultantes de defeito transientes, mas não protege contra erros de defeitos permanentes.

Pode-se incrementar o sistema de forma que quando o erro for detectado, existirem duas condições a serem analisadas:

a) Um defeito permanente produziu o erro, e aquela parte do sistema deve ser isolada (reconfigurada);
b) Um defeito transiente aconteceu e produziu o erro, mas o hardware continua sendo utilizável, sem necessidade de isolar parte do sistema.

Isto pode ser realizado em função do número de computações repetidas após a detecção do primeiro erro. (Depende dos requisitos do sistema).

5.3.2. Detecção do Defeito Permanente.

Um dos grandes potenciais da técnica da redundância temporal é a habilidade de detectar defeitos permanentes usando um mínimo de hardware extra.

Quatro abordagens são consideradas:

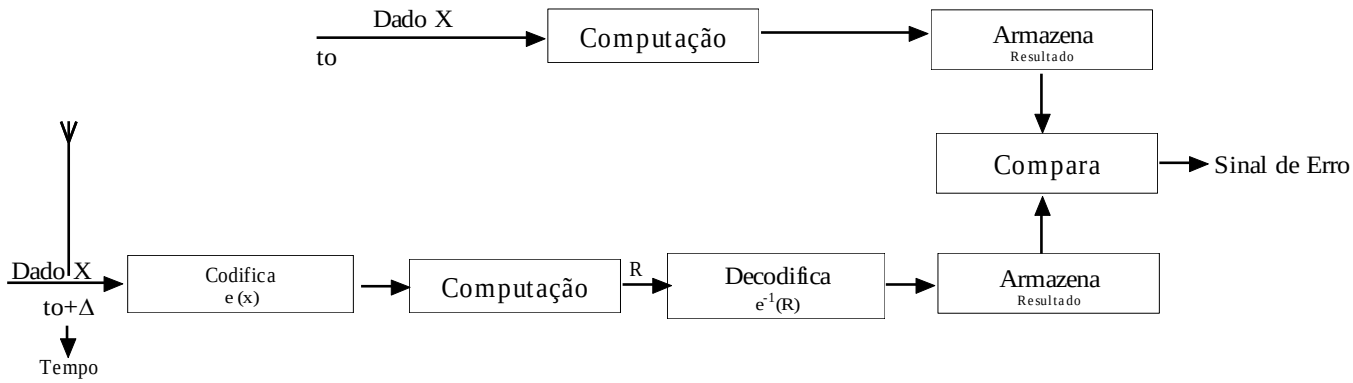
* Lógica Alternada;

* Recomputação com Operandos Deslocados;

(Recomputing with Shifted Operands - Reso);

* Recomputing with Swapped Operands (RESWO)
Recomputação com Operandos trocados
* Recomputação com Duplicação com Comparação. Recomputing with duplication with comparison (REDWC).

O conceito fundamental por trás de cada abordagem é apresentado a seguir:

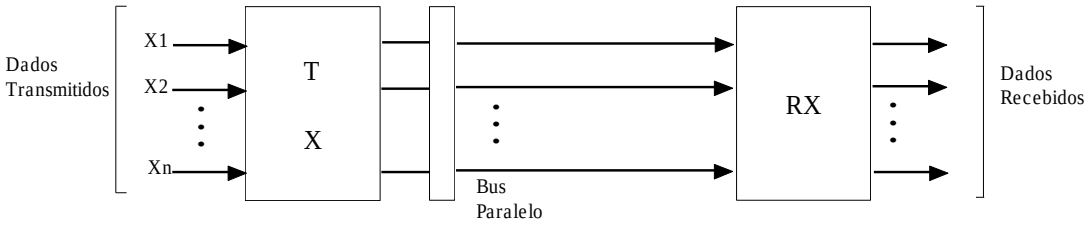


A seleção da função de codificação e (x) é feita para permitir que defeitos no hardware sejam detectados.

5.3.2.1. Lógica Alternada

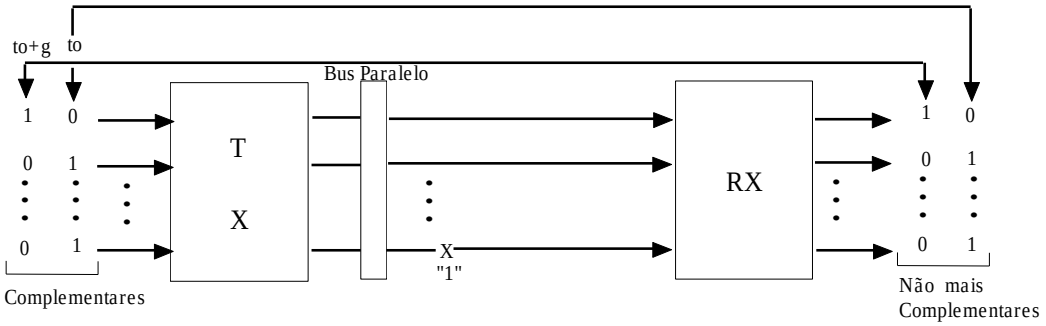
Aplicada na transmissão de dados digitais em cabos e detecção de defeitos em circuitos digitais.

Seja a figura a seguir, de transmissão de dados sobre uma via paralela.



No instante to transmite-se os dados originais e no instante to+g transmite-se os dados complementados.

Suponha um defeito numa linha, grampeando-a em nível "1".



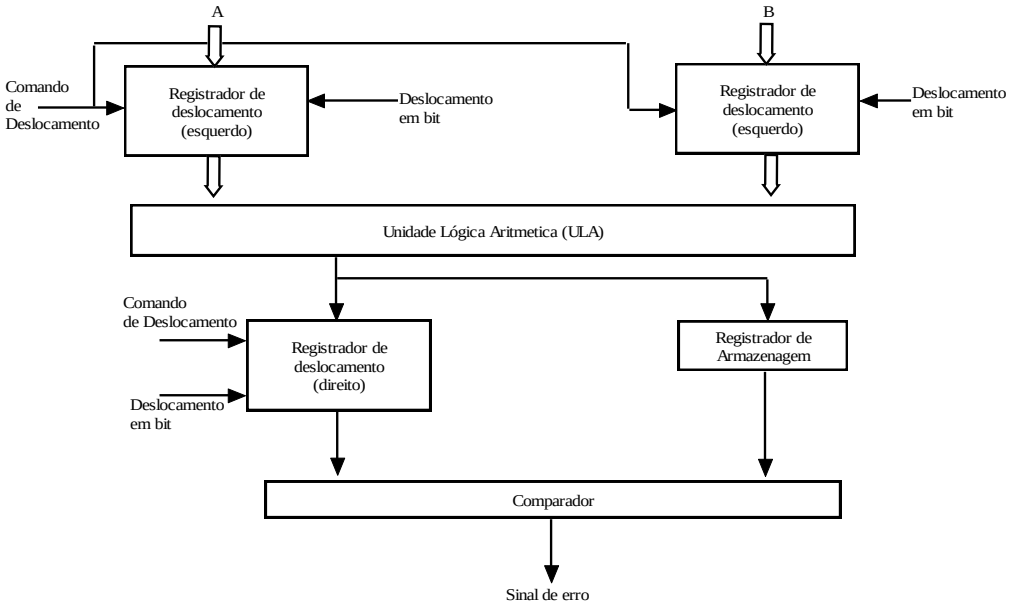
O Defeito pode ser detectado, alternando a lógica:

O conceito da Lógica Alternada pode ser aplicado, em geral, a circuitos lógicos que tem a propriedade de "self-duality", ou seja:

$f(x) = - f(-x)$

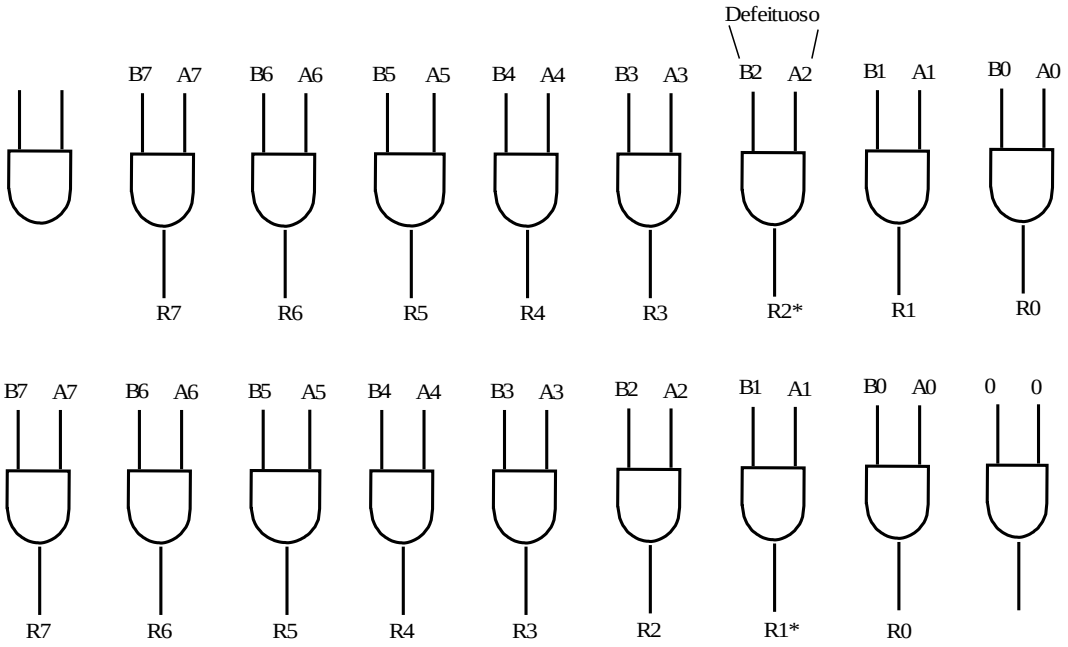
5.3.2.2. Recomputação com Operandos Deslocados

"Recomputing with Shifted Operands-RESO". Utilizado como método de fornecer detecção de erro em ULA.



A função de Codificação é selecionada como operação de deslocamento para a esquerda e a de decodificação para a direita. (deslocamento lógico ou aritmético).

Como exemplo, considere a operação lógica AND realizada sobre 8 operandos, como mostrado a seguir. Se um "bit slice" está como defeito e não tem efeito sobre os demais, um único deslocamento para à esquerda detecta o erro que ocorre na operação lógica.



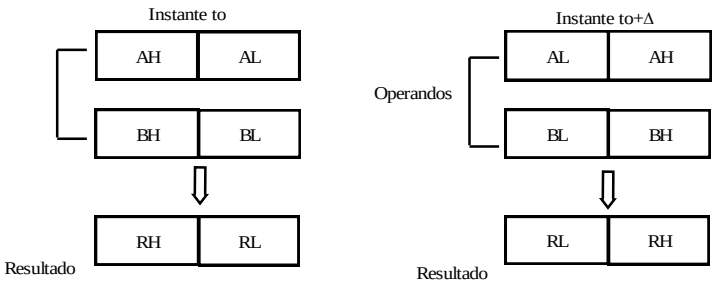
Comparação dos	R7	R6	R5	R4	R3	R2*	R1	R0
Resultados	R7	R6	R5	R4	R3	R2	R1*	R0

A estrutura de ULA que usa esta técnica é apresentada a seguir. O Hardware adicional são 3 Deslocadores, um Registrador para armazenar o resultado da primeira computação e um comparador.

O problemas primários com esta arquitetura são o hardware adicional necessário, a falta de cobertura para defeitos nos deslocadores, e o requisito que o comparador seja totalmente auto-verificável de maneira que defeitos no comparador não comprometa a eficiência da abordagem.

5.3.2.3. Recomputação com Operando Trocados

RESWO - "Recomputing with Swapped Operands". O conceito básico desta arquitetura é mostrado a seguir:



Durante a Segunda computação, a metade superior e inferior são trocadas de forma que o bit slice com defeito atua nas duas metades durante a primeira e a Segunda computação.

A estrutura da ULA projetada para acomodar a arquitetura RESWO é mostrada

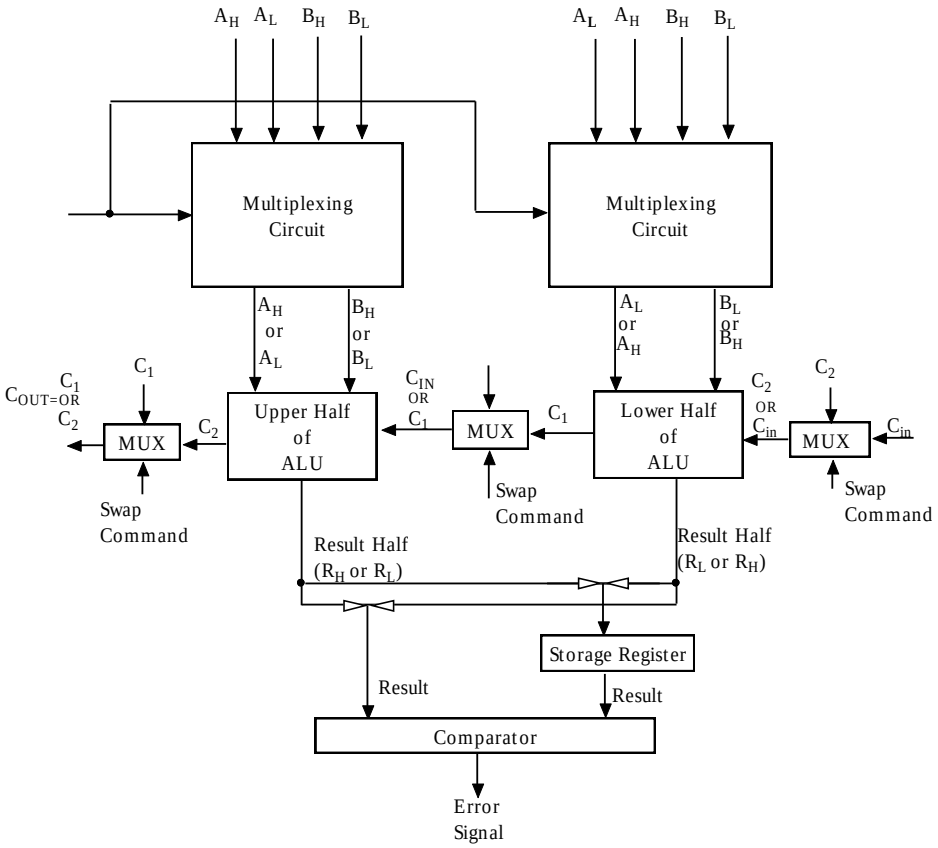
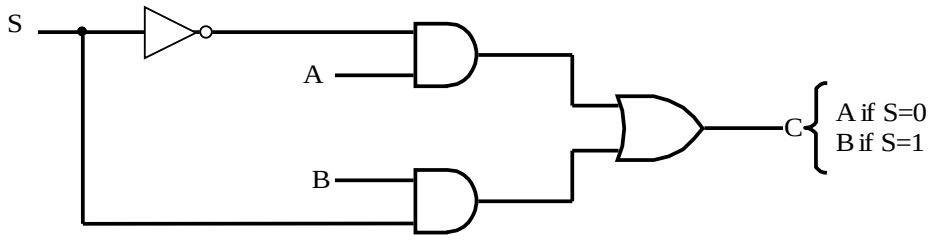


Figura 5.64

a seguir:

O multiplex é um circuito relativamente simples apresentado a seguir:



S = Swap Comand
A, B } Inputs

Figura 5.65

5.3.2.4. Recomputação com Duplicação e Comparação “Recomputing with Duplication with Compararison - REDWC”

A Redundância no tempo é utilizado para completar o cálculo e obter o resultado final.

Esta arquitetura, no caso do somador, realiza duas computações. Na primeira as metades inferiores dos operandos são somadas nas duas metades do somador. Os resultados são então comparados e um resultado é armazenado para representar a metade inferior.

A Segunda computação realiza o mesmo para as metades superiores.

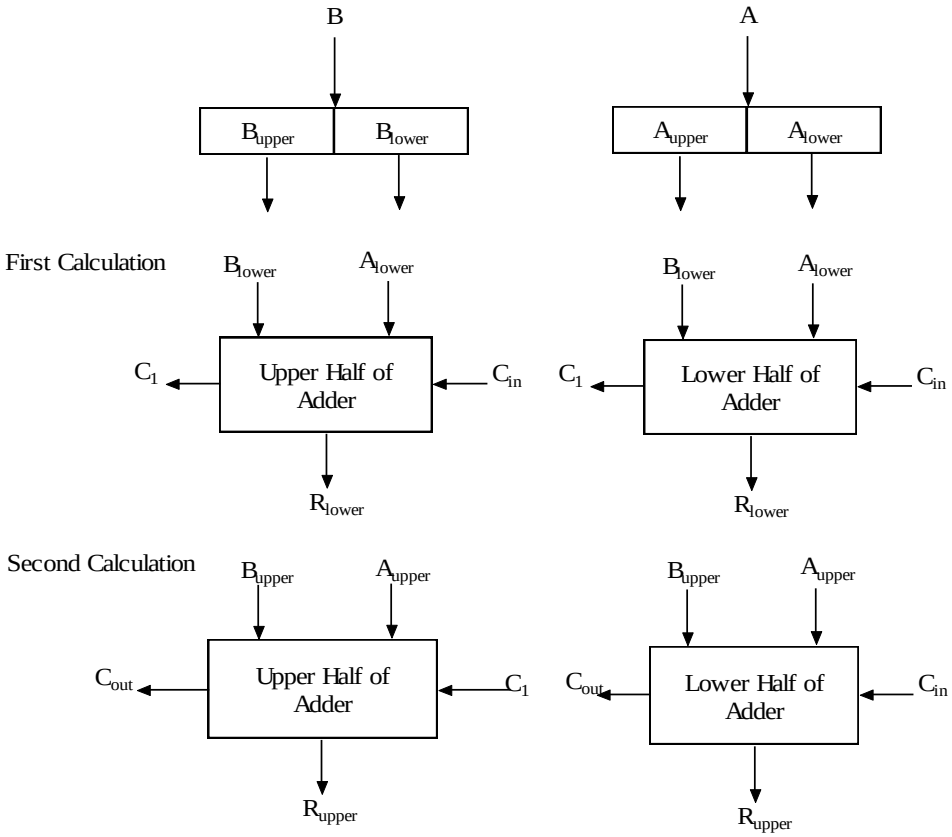


Figura 5.66

O diagrama em bloco desta arquitetura é apresentado na figura que se segue:

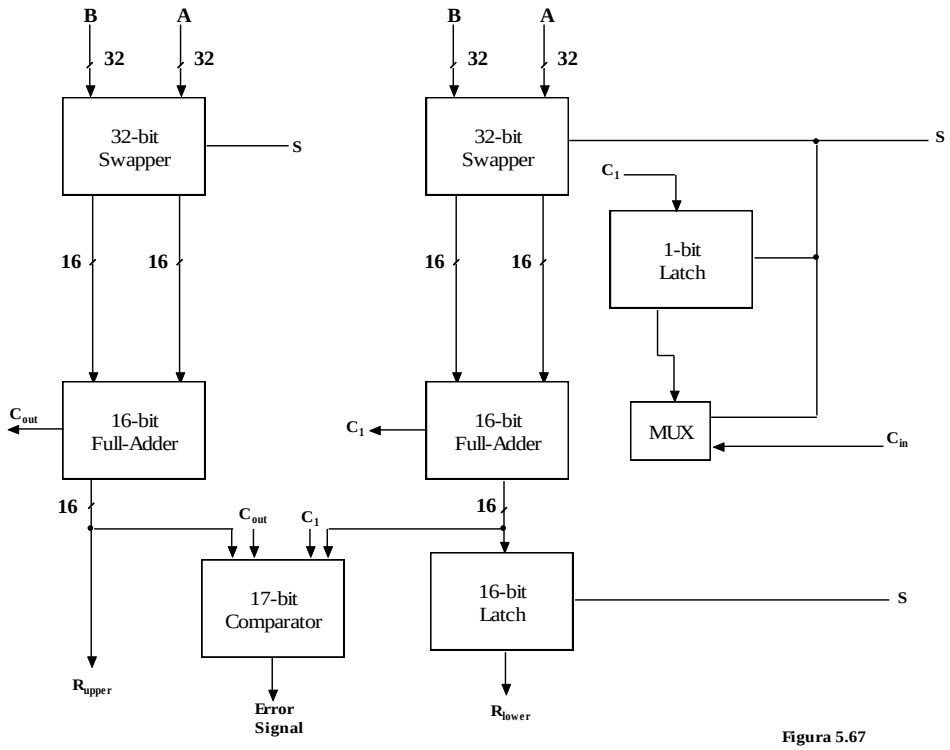
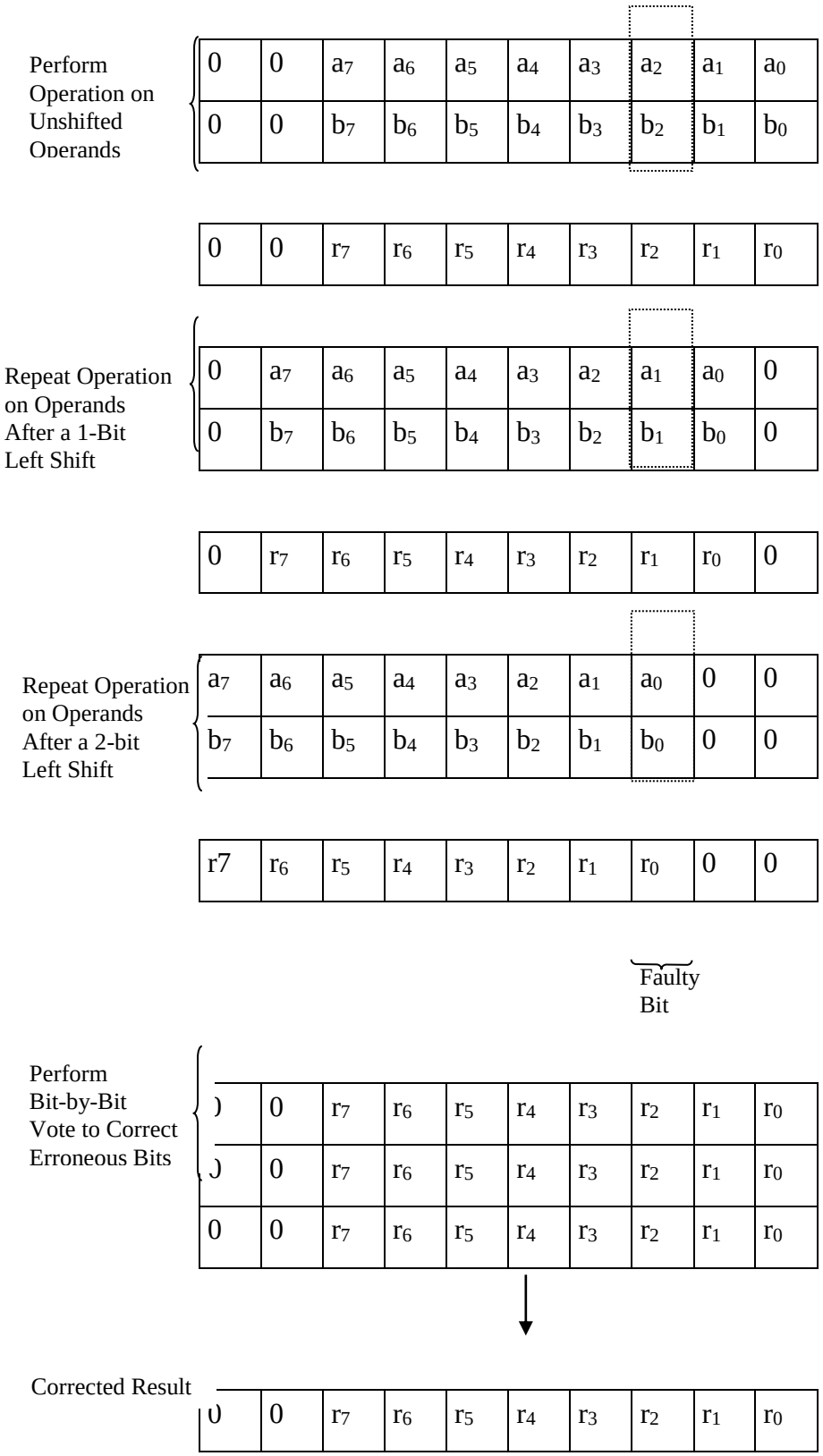


Figura 5.67

5.3.2.5. Recomputação para Correção de Erro

A Redundância pro tempo pode ser utilizada também para correção de erro se repetida 3 ou mais vezes.

Considere a operação lógica AND ilustrada na figura que se segue:


Perform Operation on Unshifted Operands
0 0 a7 a6 a5 a4 a3 a2 a1 a0
0 0 b7 b6 b5 b4 b3 b2 b1 b0
0 0 r7 r6 r5 r4 r3 r2 r1 r0
Repeat Operation on Operands After a 1-Bit Left Shift
0 a7 a6 a5 a4 a3 a2 a1 a0 0
0 b7 b6 b5 b4 b3 b2 b1 b0 0
0 r7 r6 r5 r4 r3 r2 r1 r0 0
Repeat Operation on Operands After a 2-bit Left Shift
a7 a6 a5 a4 a3 a2 a1 a0 0 0
b7 b6 b5 b4 b3 b2 b1 b0 0 0
r7 r6 r5 r4 r3 r2 r1 r0 0 0
Faulty Bit
Perform Bit-by-Bit Vote to Correct Erroneous Bits
0 0 r7 r6 r5 r4 r3 r2 r1 r0
0 0 r7 r6 r5 r4 r3 r2 r1 r0
0 0 r7 r6 r5 r4 r3 r2 r1 r0
Corrected Result
0 0 r7 r6 r5 r4 r3 r2 r1 r0

Suponha que a operação é realizada três vezes: a primeira, sem deslocamento dos operandos; a Segunda com o deslocamento de 1 bit dos operandos; a terceira, com deslocamento de 2 bits dos operandos.

Desta forma, um bit diferente no resultado será afetado pelo defeito no "bit slice". Se os bits em cada posição são comparados, o resultado devido ao defeito pode ser corrigido se realizado uma votação majoritária.

Este solução não funciona para operações aritméticas, pois os bits adjacentes não são independentes. Um defeito em um "bit slice" pode afetar mais de um bit no resultado final.

5.4. Redundância por Software

Em aplicações que utilizam computadores, muitos meios de detecção de defeitos e técnica de tolerância á defeitos podem ser implementados por software. O hardware redundante necessário para implementar esta capacidade pode ser mínimo, mas o software redundante é bastante substancial.

A Redundância de Software pode aparecer como muitas linhas extras do código usadas para verificar a magnitude de um sinal ou uma pequena rotina usada para testar periodicamente a memória (escrevendo e lendo em posições específicas).

São consideradas aqui, três técnicas de redundância de software:

- Verificação de Consistência;
- Verificação de Capacidade;
- N_ Versões de software.

5.4.1. Verificação de Consistência

A verificação de consistência usa o conhecimento a priori sobre as características da informação a ser verificada a correção. Por exemplo, em algumas aplicações, é conhecido que uma grandeza nunca deve exceder certa magnitude. Se o sinal exceder esta magnitude, indica a presença de algum erro. A verificação da consistência pode ser implementada em hardware, mas é mais realizada em software.

Um exemplo de verificação de consistência que pode ser realizado em hardware é a deteção de códigos de instrução inválidos em computadores. Muitos computadores usam n bits para representar 2^k instruções possíveis,

onde K<n. Em outras palavras, há 2^n-2^k instruções possíveis, ilegais. Cada instrução pode ser verificada que não se trata de um código ilegal. Se um código ilegal ocorre, o processador pode ser "parado" evitando operação errada de ocorrer. Esta técnica detecta erro no processador de interpretar dados como instrução.

Outro exemplo é na transferência de pacotes de dados. No início de cada pacote pode haver um "header" que contém o número de palavras naquele pacote.

Na recepção é detectado um desacordo entre o número de palavras realmente recebidas e o número especificado no "header".

5.4.2. Verificação de Capacidade

Esta verificação constata ou não que o sistema possui uma capacidade esperada.

Um primeiro exemplo é o teste de memória simples. O processador pode simplesmente escrever padrões específicos em posições de memória e le-los para verificar que o dado armazenado é recuperado. Este teste pode ser um complemento ao bit de paridade contra defeitos na memória.

Outro teste é da ULA. Periodicamente o processador executa certas instruções em dados específicos e compara os resultados com resultados esperados armazenados em ROM.

Este tipo de teste verifica tanto a ULA como a memória em que os resultados foram armazenados. As instruções executadas podem consistir de adição, multiplicação, operações lógicas e transferência de dados.

Outro tipo desta verificação consiste na verificação se todos os processadores estão se comunicando corretamente. Neste método são enviadas periodicamente informações específicas de um processador para outro.

5.4.3. N_ Versões de Software

Defeitos no software são conceitos não usuais. Técnicas para detectar estes defeitos devem detectar desvios no projeto. Uma simples duplicação e procedimento de comparação não detectam defeitos no software se os módulos de software duplicados são idênticos.

O conceito básico em N_ Versões de software é projetar e codificar o software N vezes e comparar os N resultados produzidos por cada um destes módulos. Cada um desses módulos é projetado e codificado por um grupo separado de programadores. Cada grupo projeta o software a partir de uma mesma especificação, de maneira que cada módulo desempenha a mesma função.

É esperado que N projetos independentes não irão gerar erros iguais.

Entretanto quando ocorre um erro, não ocorre em todos os módulos, ou ocorre de maneira diferente em cada módulo, de forma que os resultados finais serão diferentes.

Existem certas polêmicas com relação à N_ Versões de software.

A primeira é que os projetistas e codificadores tendem a fazer erros similares.

Além disso, não se garante que versões independentes de um programa não terão defeitos idênticos.

Finalizando, como as N versões se originam a partir de uma mesma especificação, não se detecta erros na especificação.

Se o software é desenvolvido corretamente, não há a necessidade de se utilizar técnicas de tolerância a defeitos de software.

6. Técnicas de Avaliação de Sistemas Tolerantes a Defeito

- Taxa de Falhas
- Tempo médio para Falhar (MTTF)
- Tempo médio entre Falhas (MTBF)
- Cobertura de Defeitos
- Análise de Confiabilidade
- Análise de Segurança
- Análise de Disponibilidade
- Análise de Manutenabilidade
- Taxas de Redundância
- Custos

6.1. Função Confiabilidade (R (t)) e Taxa de Falhas

Para se definir a Função Confiabilidade *R(t)* é necessário conceituar o termo taxa de falhas. Intuitivamente a taxa de falha é o número esperado de falhas de um tipo de sistema num determinado período de tempo.

Como exemplo, pode-se considerar que um computador falha, em média, uma vez a cada 2000 horas. Neste caso o computador apresentará uma taxa de falhas de 1/2000 falhas/hora.

A taxa de falha é denotada pela letra grega λ.

Quando algum tipo de redundância for incorporado como meio de alcançar tolerância a defeito, a taxa de falhas desse novo sistema redundante deve ser menor do que a taxa da falha do sistema similar, não redundante.

Para podermos representar formalmente o conceito de Confiabilidade pode-se seguir o seguinte raciocínio.

Vamos utilizar *N* componentes idênticos.

Suponha que estão todos operacionais no instante *to* e que se registre o número de componentes falhos e componentes corretos no instante *t*.

$N_f(t)$: Número de componentes que falharam até o instante *t*.

$N_o(t)$: Número de componentes operacionais até o instante *t*.

A confiabilidade *R(t)* neste exemplo pode ser calculada como a divisão entre o número de componentes operacionais, dividido pelo número total de componentes.

$$R(t) = \frac{N_o(t)}{N} = \frac{N_o(t)}{N_o(t) + N_f(t)}$$

Esse valor corresponde a probabilidade de um componente funcionar no intervalo entre *[to,t]*.

Por outro lado, a Não-Confiabilidade *Q(t)* de um componente funcionar no intervalo *[to,t]* pode ser calculada como:

$$Q(t) = \frac{N_f(t)}{N} = \frac{N_f(t)}{N_o(t) + N_f(t)}$$

Desta forma:

$$R(t) = 1 - Q(t) = 1 - \frac{N_f(t)}{N}$$

$$\frac{dR(t)}{dt} = -\frac{1}{N} . \frac{dN_f(t)}{dt}$$

$\frac{dN_f(t)}{dt} = -N . \frac{dR(t)}{dt}$, considerada como taxa de falha instantânea.

A Função Taxa de Falha $\lambda(t)$ ou "*Hazard Function*" ou "*Hazard Rate*" é definida como:

$$\lambda(t) = \frac{\frac{dN_f(t)}{dt}}{N_o(t)} \qquad \text{(falhas por unidade de tempo)}$$

$$\lambda(t) = \frac{1}{N_o(t)} . (-N . \frac{dR(t)}{dt}) = -\frac{N}{N_o(t)} . \frac{dR(t)}{dt}$$

$$\lambda(t) = -\frac{\frac{dR(t)}{dt}}{R(t)} = \frac{\frac{dQ(t)}{dt}}{1-Q(t)}$$

Sendo que $f(t) = \frac{dQ(t)}{dt}$ correspondente à função densidade de falha.

A função taxa de falhas *λ(t)* depende do tempo, e verifica-se experimentalmente que para componentes eletrônicos e computacionais existe um período em que *λ(t)* é aproximadamente constante. Este fato está relacionado com a denominada Curva da Banheira, conforme mostra a figura 6.1.

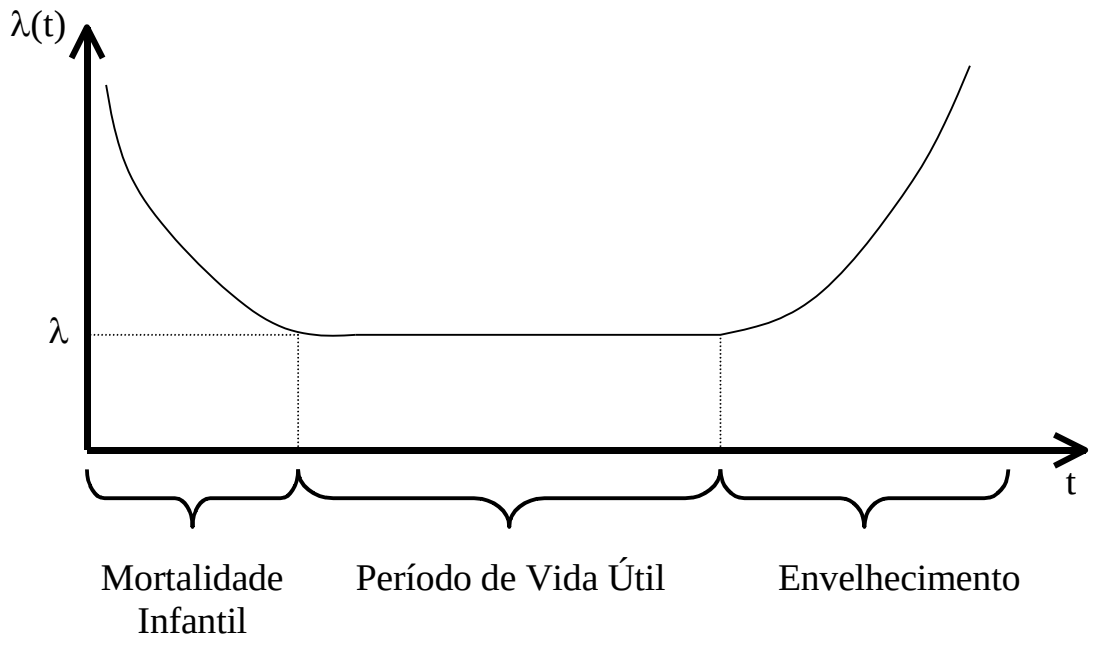


Figura 6.1 – Curva da Banheira

A região de Mortalidade Infantil corresponde à região em que aqueles componentes produzidos com algum defeito constitucional falham. Para se eliminar esses componentes com uma maior rapidez, pode-se utilizar técnicas de Teste de Burn-in, onde os componentes são submetidos a um stress maior do que aquele em que é submetido quando em operação normal. O Período de Vida Útil corresponde ao período de tempo em que a taxa de falhas do

componente se mantém relativamente estável. O período de envelhecimento corresponde ao período em que a taxa de falhas do componente começa a crescer vertiginosamente, aumentando drasticamente a probabilidade do componente falhar, caso não tenha ainda falhado.

Já havíamos concluído que:

$$\lambda(t) = -\frac{\frac{dR(t)}{dt}}{R(t)}$$

$$\frac{dR(t)}{dt} = -\lambda(t).R(t)$$

Se admitirmos que o sistema está no período de vida útil, a função taxa de falha tem um valor constante, denominado λ.

$$\frac{dR(t)}{dt} = -\lambda..R(t)$$

A solução para essa equação é:

$$R(t) = e^{-\lambda.t}$$

que corresponde a uma curva de distribuição exponencial. Vale ressaltar que essa expressão só é válida para componentes eletrônicos e computacionais.

Em certas aplicações, entretanto, não se pode assumir que a função de taxa de falha apresenta valor constante. Entre estas aplicações pode-se citar sistemas mecânicos, pneumáticos, hidráulicos e o próprio componente Software. Nestes casos outros modelos devem ser empregados.

6.2. Cálculo da Taxa de Falhas

Um aspecto importante na análise de sistema é a estimativa de taxa de falhas de componentes específicos. A técnica mais comum é do Departamento de Defesa Americano (USDOD) cuja norma MIL - HDBK - 217 contém esses dados a partir de valores experimentais.

Apresenta - se a seguir alguns parâmetros.

A medição de falha de um CI é dada por:

$\lambda = \pi L . \pi Q (C1 . \pi_t + C2 . \pi_E) . \pi_P$ falhas por milhões de horas.

πL... Fator de aprendizado: representa a maturidade do processo do fabricante utilizado na produção do CI (Varia de 1 a 1ϕ)

πQ ... Fator de qualidade: representa o conjunto de teste que o componente sofre antes de ser vendido por um fornecedor (1 a 3$\phi\phi$)

Existem 4 níveis básicos: A, B, C, D.

Classes A e B => aplicações militares

A => $\pi Q = 1$

B => $\pi Q = 2$

Classe C => componentes comerciais de alta qualidade.

$\pi Q = 16$

Classe D => componentes comerciais hermeticamente selados.

$\pi Q = 150$

πT... Fator de temperatura. É função da tecnologia, temperatura de operação, tecnologia de empacotamento e dissipação de potência. (0, 1 a 1000).

πE... Fator ambiental. Baseado no ambiente operacional.

Ex.: Componentes em sala de computador - 0, 2

componentes em mísseis - 10, 0

πp ... Função do No de pinos do CI.

1, 0 => No de pinos $\leq$ 25

1, 1 => 26 < n < 64

1, 2 => n > 64

C1, C2 ... Fator de Complexidade; função do número de portas em circuito lógicos, número de transistores para circuitos lineares e No de bits para memórias.

Number of logic gates	Failure rate (Failures per milion hours)
(a) logic circuits	
50	0.1527
100	0.2312
200	0.3655
500	1.4483
1000	14.4880

Memories (RAM) Number of bits	Failure rate (Failures per milion hours)
1024 (1K)	0.8837
2048 (2K)	1.3491
8192 (8K)	3.1453
16.384 (16K)	4.8033
32.768 (32K)	7.3362

6.3 Tempo Médio para Falhar – "Mean Time to Failure" – MTTF e a Confiabilidade de um Sistema

O Tempo Médio para Falhar - *MTTF* corresponde ao tempo médio esperado em que o sistema irá operar antes que a primeira falha ocorra.

Num exemplo prático, se tivermos *N* sistemas idênticos colocados em operação no instante t = 0 e medirmos o tempo que cada um desses *N* sistemas operem antes da ocorrência de uma falha, pode-se determinar o *MTTF* como:

$$MTTF = \frac{\sum_{i=1}^{N} t_i}{N}$$

Dada teoria de probabilidade, pode-se determinar o valor médio de uma variável *x* em função de sua densidade de probabilidade *f(x)* através da seguinte formulação:

$$E[x] = \int_{-\infty}^{+\infty} x.f(x).dx$$

Considerando-se na expressão anterior a variável *x* como sendo *tempo operacional até uma falha*, e a função *f(x)* como a *função de densidade de falha*, o valor de *E(x)* corresponderá ao valor de *MTTF*. Desta forma têm-se:

$$MTTF = \int_{0}^{+\infty} t.f(t).dt$$

Como trata-se da variável tempo, não tem sentido valores negativos, razão pela qual a integral é realizada nos intervalos de 0 a ∞.

A *função densidade de falha f(t)* pode ser calculada através da variação da Não - Confiabilidade:

$$f(t) = \frac{dQ(t)}{dt}$$

Substituindo a função *f(t)* na expressão do *MTTF* têm-se:

$$MTTF = \int_{0}^{\infty} t.\frac{dQ(t)}{dt}.dt = -\int_{0}^{\infty} t.\frac{dR(t)}{dt}.dt$$

O principal objetivo neste momento é procurar obter uma relação entre a confiabilidade *R(t)* e o *MTTF*. Para tal, utilizaremos a teoria de cálculo integral.

Dada duas funções, pode-se calcular a derivada da multiplicação através da seguinte formulação:

$$\frac{d\,f(t).g(t)}{dt} = f(t).\frac{dg(t)}{dt} + g(t).\frac{df(t)}{dt}$$

Realizando-se a integração entre os instante *0* e *t* de ambos os lados têm-se:

$$\int_{0}^{t} \frac{df(t).g(t)}{dt}.dt = \int_{0}^{t} f(t).\frac{dg(t)}{dt}.dt + \int_{0}^{t} g(t).\frac{df(t)}{dt}.dt$$

Pode-se então escrever:

$$[f(t).g(t)]_0^t = \int_0^t f(t).\frac{dg(t)}{dt}.dt + \int_0^t g(t).\frac{df(t)}{dt}.dt$$

Considerando *f(t) = R(t)* e *g(t) = t* têm-se:

$$[R(t).t]_0^t = \int_0^t R(t).dt + \int_0^t t.\frac{dR(t)}{dt}.dt$$

Quando *t* tende a infinito pode-se escrever:

$$[R(t).t]_0^\infty = \int_0^\infty R(t).dt - MTTF$$

Como pode considerar que o valor de *R(t)* tende a zero antes que *t* tenda a infinito, pode-se afirmar que:

$$(R(t).t)^{t\to\infty} - (R(t).t)^{t\to 0} = 0$$

Assim têm-se:

$$0 = \int_0^\infty R(t).dt - MTTF$$

Desta forma pode-se concluir uma expressão geral que represente o valor de *MTTF* em função da Confiabilidade.

$$MTTF = \int_0^\infty R(t).dt$$

Caso a função confiabilidade *R(t)* obedeça à lei de falha exponencial, característica plausível para componentes elétricos/eletrônicos/computacionais, têm-se:

$$MTTF = \int_0^\infty e^{-\lambda.t}.dt = [\frac{e^{-\lambda.t}}{-\lambda}]_0^\infty = \frac{1}{\lambda}$$

Pode-se concluir que o valor do *MTTF*, para componentes que apresentem uma função de confiabilidade exponencial, corresponde ao valor inverso da taxa de falhas.

Se formos calcular a confiabilidade de um sistema no instante $t = MTTF$, com distribuição exponencial de falhas, tem-se:

$$R(MTTF) = e^{-\lambda.MTTF} = e^{-\lambda.\frac{1}{\lambda}} = e^{-1} = 0{,}3678$$

Em outras palavras um sistema obedecendo a lei de falha exponencial tem uma probabilidade de 0,3678 de não experimentar uma falha antes do tempo igual a *MTTF*, dado que o sistema estava em correto funcionamento no início do período de tempo.

6.4. Tempo Médio para Reparo

O Tempo Médio para Reparo – *MTTR* corresponde simplesmente ao tempo médio requerido para reparar um determinado sistema, que se encontrava falho.

Neste sentido, pode-se avaliar este tempo inserindo-se *N* defeitos em *N* sistemas idênticos e medindo-se o tempo que a equipe de manutenção leva para repará-los. Evidentemente este tempo de reparo irá depender da qualidade do alarme sendo emitido, destacando sua precisão quanto à localização exata da falha origem. Outro aspecto importante é que os *N* defeitos introduzidos sejam representativos dos possíveis de acontecerem, podendo ser considerados como uma boa representação do universo de defeitos possíveis. Neste livro não se entrará em maiores detalhes sobre a representativa dessa amostra e do grau de confiança do resultado final. O objetivo principal é se chegar a uma expressão geral do conceito de *MTTR*.

Tendo feito essas considerações pode-se considerar o *MTTR* como:

$$MTTR = \frac{\sum_{i=1}^{N} t_i}{N}$$

Quando se trata de uma manutenção em laboratório, em que a rapidez e a qualidade da manutenção dependem muito do nível técnico do profissional envolvido na atividade, este valor do *MTTR* pode ser extremamente influenciado pelo tipo de profissional e, portanto, deve ser levado em consideração. Tendo em vista este novo aspecto e considerando *M* técnicos envolvidos no trabalho, o novo *MTTR* pode ser calculado como:

$$MTTR = \frac{\sum_{i=1}^{M} MTTR_i}{M}$$

onde $MTTR_i$ corresponde ao valor médio calculado anteriormente sobre *N* tipos de defeitos inseridos.

O *MTTR* pode também ser representado através do inverso da taxa de reparo *(μ)*, que é o número médio de reparos que ocorrem num determinado período de tempo.

6.5. Tempo Médio Entre Falhas

O Tempo Médio entre Falhas - *MTBF* pode ser calculado através da media de tempo entre falhas consecutivas, incluindo o tempo para reparar o sistema e colocá-lo de volta ao estado operacional.

Para a realização prática desta medida pode-se colocar *N* sistemas idênticos em funcionamento por um período de tempo *T*, e anotar o número total de falhas em cada um desses sistemas *(ni)* considerando-se, evidentemente, a realização de manutenções.

Desta forma pode-se estabelecer que o número médio de falhas seja avaliado como:

$$N_{Médio} = \frac{\sum_{i=1}^{N} n_i}{N}$$

Como esse número médio de falhas foi detectado ao longo de um período de tempo *T*, pode-se calcular o *MTBF* como:

$$MTBF = \frac{T}{N_{Médio}}$$

Para se compreender a relação entre o *MTBF*, o *MTTF* e o *MTTR* pode-se observar a figura 6.2, que ilustra os seus instantes de ocorrência.

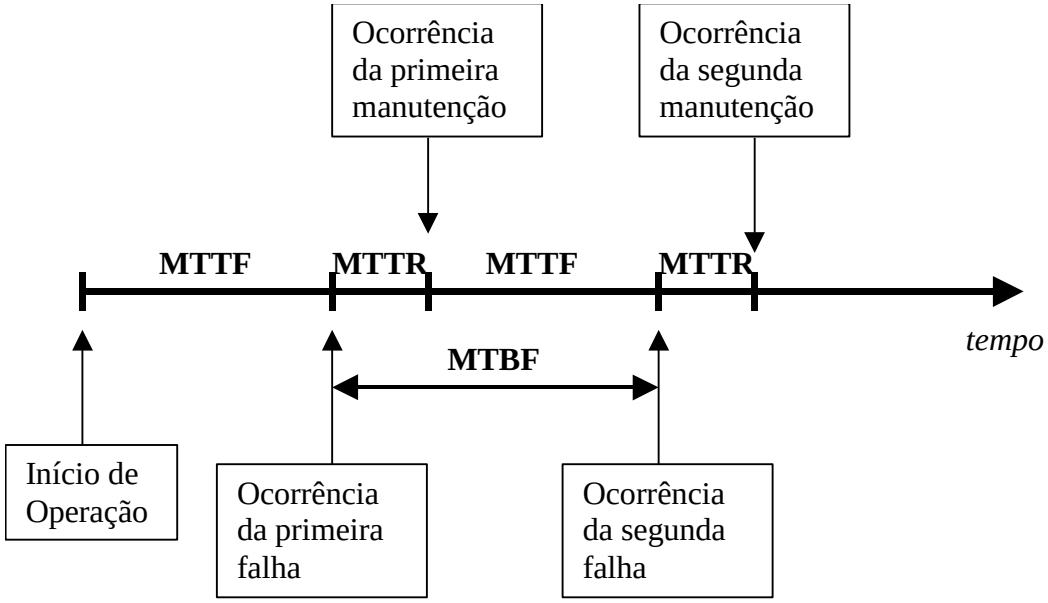


Figura 6.2 – Relação entre MTTUF, MTTR e MTBF

Com base na figura pode-se concluir que:

$$MTBF = MTTF + MTTR$$

Na maioria dos sistemas o valor do *MTTR* corresponde a uma pequena fração do valor do *MTBF*. Este fato torna os valores do *MTTF* e do *MTBF* bastante próximos. Do ponto de vista conceitual eles são valores bastantes diferentes, diferença essa que irá influenciar na determinação da disponibilidade final do sistema.

6.6 A Disponibilidade Assintótica de um Sistema

Conforme já foi comentado anteriormente, a disponibilidade de um sistema corresponde à probabilidade dele estar funcionando corretamente em um determinado instante de tempo.

No entanto, esta curva de disponibilidade tende a um valor estável, denominado disponibilidade assintótica.

Este valor de disponibilidade assintótica pode ser calculado da seguinte forma:

$$A_{assintótica} = \frac{TempoOperacional}{TempoTotal} = \frac{\sum_{i=1}^{n}(MTTF_i)}{\sum_{i=1}^{n}(MTTF_i + MTTR_i)}$$

Dessa forma, têm-se:

$$A_{assintótica} = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$$

Por exemplo, um sistema computacional com *MTTF* igual a 10.000 horas e um valor de *MTTR* igual a 1 hora, apresenta uma disponibilidade assintótica de 0,9999.

6.7. Cobertura de Defeitos.

A Cobertura de Defeito num sistema pode ter um tremendo impacto na Confiabilidade, Segurança e outros atributos do sistema.

Há muitos tipos de cobertura, dependendo se o projetista está querendo detecção do defeito, localização do defeito, (local e não global) contenção do defeito ou recuperação (manter o estado operacional) do defeito.

A Cobertura da Detecção do Defeito é a medida da habilidade do sistema em detectar defeitos e assim nos outros casos.

Na maioria dos casos, “cobertura de defeito” implica em “Cobertura da Recuperação do Defeito”.

Matematicamente, é a probabilidade condicional dada a existência de um defeito, do sistema recuperar:

C = P (fault recovery / fault existence).

A cobertura da Recuperação dos Defeitos é calculada como a função das falhas, que podem ser recuperadas, dividida pelo número total de falhas.

Exemplo:

Seja o circuito a seguir: (15 pontos de erro).

preso em “0”

preso em “1”

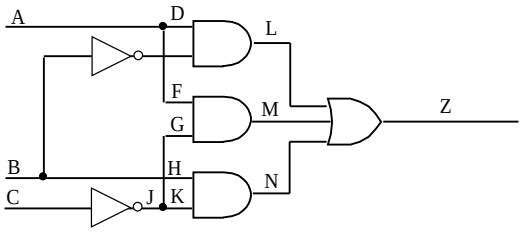


Figura 5.79

Figura 6.3

Como resultado, tem-se que a cobertura da detecção da falha é:

C = 30 - 3 (N° de modos de falhas detetáveis) = 0,9 ⟶ 90%

30 (N° de modos de falhas possíveis)

A tabela a seguir, apresenta todos os defeitos possíveis e seus respectivos vetores de teste de detecção.

Fault	Number of test vectors	Test vectors ABC
A_0	2	100, 101
A_1	2	000, 001
B_0	2	010, 111
B_1	2	000, 101
C_0	2	011,111
C_1	2	010, 110
D_0	1	101
D_1	2	000, 001
E_0	1	101
E_1	1	111
F_0	0	
F_1	2	000,001
G_0	0	
G_1	1	111
H_0	1	010
H_1	1	000
I_0	1	111
I_1	1	101
J_0	2	010,110
J_1	2	011,111
K_0	1	010
K_1	2	011,111
L_0	1	101
L_1	4	000,001,011,111
M_0	0	
M_1	4	000,001,011,111
N_0	1	010
N_1	4	000,001,011,111
Z_0	4	010,100,101,110
Z_1	4	000,001,011,111

6.8. Modelo de Confiabilidade

Um problema de se medir este atributo de um sistema é o número de sistemas necessários para se atingir um resultado bom. Isto é problemático quando existe a limitação do número de sistemas.

Outro problema é que os sistemas tem atingido índice de confiabilidade de $0, 9_7$ depois de 10 horas de operação, que corresponde a uma taxa de falhas, de 10^{-8} falhas/h.

$R(t) = e^{-\lambda t} = 0{,}9999999$

$\ln(0, 9999999) = -\lambda . 10 = -10^{-7}$ falhas/h

$\lambda = 10^{-8}$

1 falhas a cada 11416 anos.

As técnicas de análise de confiabilidade mais usuais são as abordagens analíticas. As que mais se destacam são os Modelos Combinatórios e por Markov.

6.8.1. Modelos Combinatórios

Esses modelos usam técnicas probabilísticas que enumeram os diferentes caminhos no qual o sistema pode permanecer operacional.

As probabilidades dos eventos que fazem com que o sistema permaneça operacional são calculadas para formar uma estimativa da confiabilidade do sistema.

A confiabilidade de um sistema é geralmente derivada em termos de confiabilidade dos componentes individuais do sistema. Os dois modelos mais comuns na prática são: **série** e **paralelo.** Num modelo **série**, cada elemento do sistema é requerido operar corretamente para que o sistema opere corretamente. No modelo **paralelo,** por outro lado, apenas um dos diversos elementos deve estar operacional para que o sistema desempenhe suas funções corretamente.

Na prática, os sistemas são típicas combinações de subsistemas série e paralelo.

Serão discutidas agora as técnicas de modelagem de sistema **série** e **paralelo.**

6.8.1.1. Sistema séries

O Sistema série é a melhor abordagem de um sistema que não contém redundância, ou seja, cada elemento do sistema é necessário para fazê-lo funcionar corretamente.

Uma maneira de representar o sistema série é através do DIAGRAMA DE BLOCO DE CONFIABILIDADE.

Este diagrama pode ser pensado como o diagrama de fluxo desde a entrada do sistema até a sua saída. Cada elemento do sistema é um bloco no diagrama de série. Os blocos são colocados em série para indicar o caminho da entrada para a saída. O fluxo é quebrado se um dos elementos falhar.

Um Diagrama de Bloco de Confiabilidade geral é apresentado a seguir é apresentado a seguir:

Figura 6.1

Figura 6.4

A confiabilidade do sistema série pode ser calculada como a probabilidade de nenhum dos elementos falharem. Outra maneira é que a confiabilidade do sistema série é a probabilidade de todos os elementos funcionarem corretamente.

Vamos supor que Ciw (t) represente o evento do componente Ci estar funcionando corretamente no instante t, e R series (t) é a confiabilidade do sistema série.

A confiabilidade do sistema série, no instante t é:

$$R_{series}(t) = P(C1W(t) \cap C2W(t) \cap \ldots \cap CNW(t))$$

Assumindo que os eventos CiW (t) são independentes:

Rseries (t) = R1 (t) .R2 (t) .R3 (t)... RN (t)

$$R_{series}(t) = \prod_{i=1}^{N} R_i(t)$$

Uma relação interessante existe num sistema série se cada componente individual satisfaz a lei de falhas exponencial.

Suponha que cada componente tenha uma taxa de falha i, e que $R_i(t) = e^{-\lambda_i t}$..

A confiabilidade do sistema será:

$$R_{series}(t) = e^{-1t} \cdot e^{-2t} \ldots e^{-\lambda_i t}$$

$$R_{series}(t) = e^{-t \cdot \sum_{i=1}^{N} \lambda_i} = e^{-\lambda_{sist} \cdot t}$$

$$\lambda_{sistema\ série} = \sum_{i=1} \lambda_i$$

que corresponde á taxa de falha do sistema.

Exemplo: (Sistema de controle de avião)

Veja o sistema apresentado a seguir:

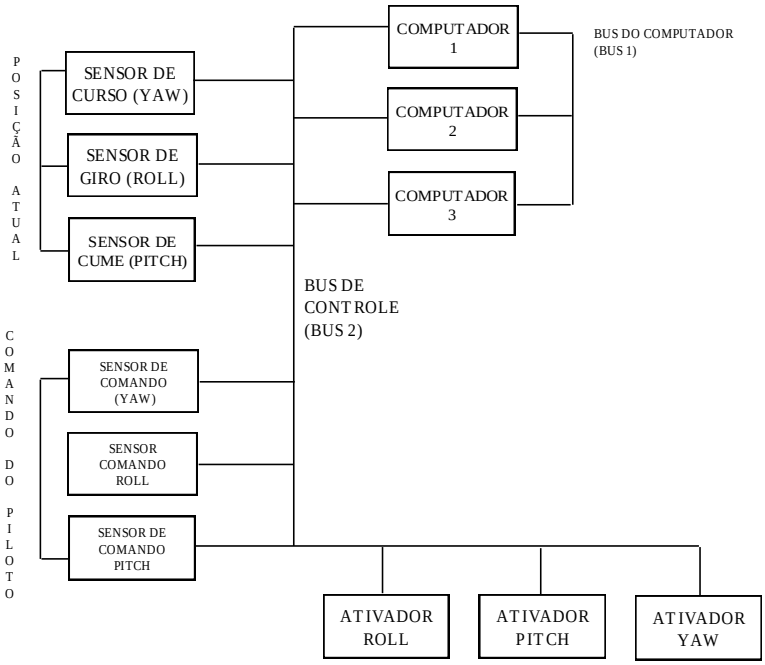


Figura 6.2

Os computadores recebem dos sensores de comando posições desejadas, comparam com as atuais, processam e enviam aos atuares através do BUS2. Um bus de alta velocidade interconecta os computadores com o propósito de transferência de dados entre os computadores.

Não há redundância, e cada elemento do sistema é requerido para que o sistema é requerido para que o sistema desempenhe corretamente sua função.

O Diagrama em bloco de Confiabilidade é apresentado a seguir:

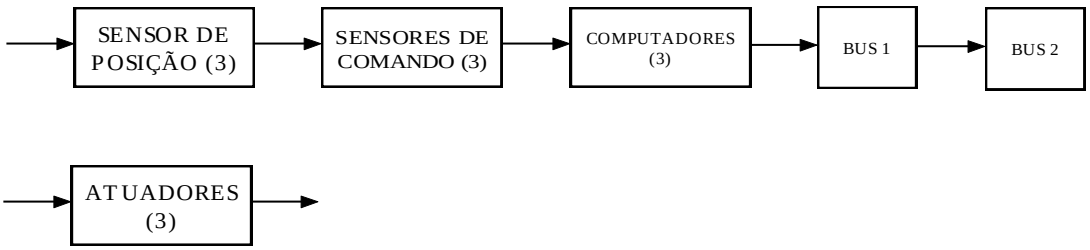


FIGURA 6.3

Figura 6.6

Por simplicidade assume-se que todos os seis sensores tem a mesma confiabilidade Rs (t) , cada atuador tem a confiabilidade Ract (t) , o bus1 tem confiabilidade $Rbus_1$ (t) e o bus $_2$ de controle $Rbus_2$ (t)

O computador tem confiabilidade Ract (t).

$$= \Rightarrow R_{sistema}(t) = R_s(t)^6 . R_{act}(t)^3 . R_c(t)^3 . R_{bus_1}(t) . R_{bus_2}(t)$$

$$= \Rightarrow \lambda_{sistema} = 6\lambda_s + 3\lambda_{act} + 3\lambda_c + \lambda_{bus1} + \lambda_{bus2}$$

taxa de falha do computador

taxa de falha do sensor

taxa de falha do atuador.

As taxas de falha de cada componente são:

$\lambda_S = 1 \times 10\text{-}6$ falhas por hora

$\lambda_{act} = 1 \times 10\text{-}5$ falhas por hora

$\lambda_c = 4 \times 10\text{-}4$ falhas por hora

$\lambda_{bus_1} = 1 \times 10\text{-}6$ falhas por hora

$\lambda_{bus_2} = 2 \times 10\text{-}6$ falhas por hora

$\lambda_{sistema} = 1{,}239 . 10\text{-}3$ falhas por hora

A confiabilidade é:

$$R_s(t) = e^{-\lambda_s . t} = e^{-1{,}239 . 10\text{-}3 . t}$$

Após 5 horas a confiabilidade é

R_s (5h) = 0,994

$$\text{O MTTFs} = \frac{1}{\lambda_s} = \text{MTTF} = 807{,}10 \text{ horas}$$

(Baixíssima)

6.8.1.2. Sistemas Paralelos

A característica básica deste sistema é que apenas um dos N elementos idênticos é requerido para o funcionamento do sistema.

O Diagrama em Bloco de Confiabilidade de um sistema paralelo, que contém N elementos idênticos, está mostrado abaixo:

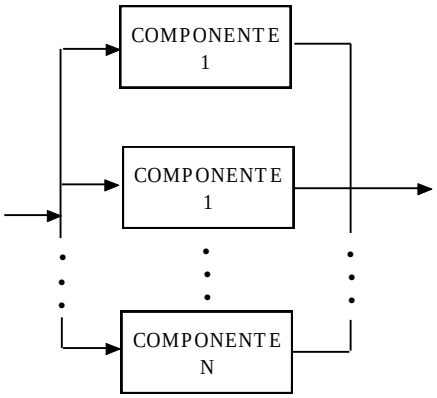


FIGURA 6.4

Figura 6.7

A não confiabilidade do sistema paralelo pode ser computada como a probabilidade de que todos os N elementos falhem.

Seja Cif (t) o evento que o elemento i tenha falhado no instante t, e Q_{paral} (t) a não confiabilidade do sistema paralelo e Qi (t) a não confiabilidade do i ésimo elemento.

$$Qparal.(t) = P\ (C1f\ (t) \cap C2f(t) \cap .... \cap\ Cnf\ (t))$$

$$Qparal.(t) = Q1\ (t).Q2(t)...Qn(t) = \prod_{i=1}\ Qi\ (t)$$

Matematicamente tem-se R(t) +Q(t) = 1

$$R(t) = 1{,}0 - Q(t) = 1{,}0 - \prod_{i=1}^{N} Qi(t) =$$

$$1{,}0 - \prod_{i=1}^{N} (1 - Ri(t))$$

Neste modelo considera-se também que as falhas dos elementos são independentes.

Para falhas de hardware a independência das falhas é uma boa suposição, mas para falhas resultantes de distúrbios externos a independência não é uma boa hipótese, pois o distúrbio externo é um evento comum, provador de diversos tipos de falhas.
Para ilustrar a aplicação do modelo seja o sistema a seguir:

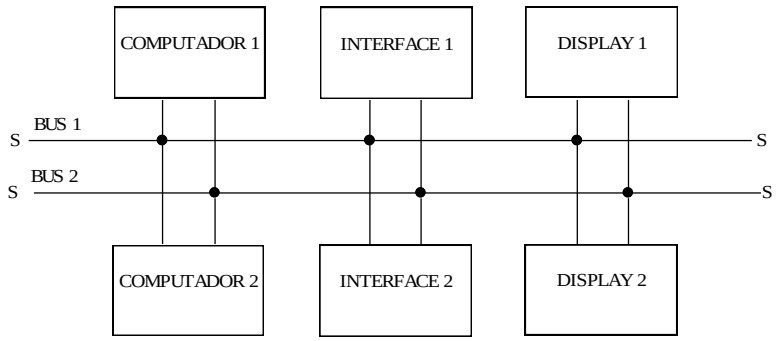


Figura 6.8

O sistema requer que pelo menos uma unidade de cada componente opere corretamente para que o sistema desempenhe suas funções.

Uma vez que uma unidade particular falhe, é assumido que a outra unidade assuma automaticamente as funções.

Um aspecto importante neste exemplo é que ela apresente estrutura paralela e serial.

O aspecto paralelo é que apenas um dos componentes (mesmo tipo) deve funcionar. O aspecto série é que um componente de cada tipo deve estar funcionando.

O Diagrama em Bloco da Confiabilidade do sistema é mostrado a seguir:

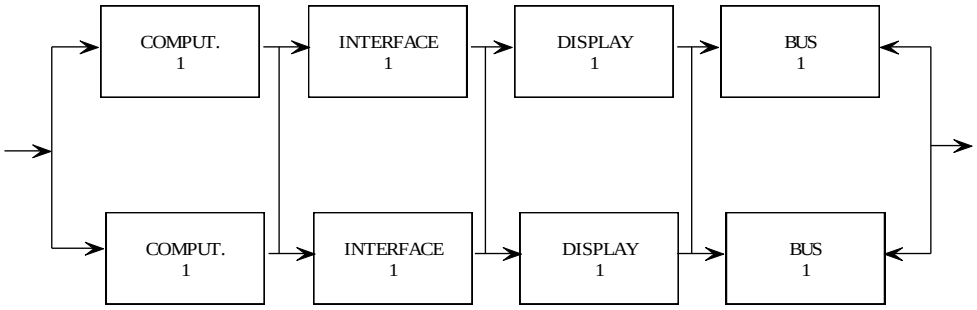


Figura 6.9

Este diagrama Série/ Paralelo pode ser reduzido a um diagrama série trocando cada porção paralela do sistema por uma equivalente, com a mesma confiabilidade.

Veja a figura a seguir:

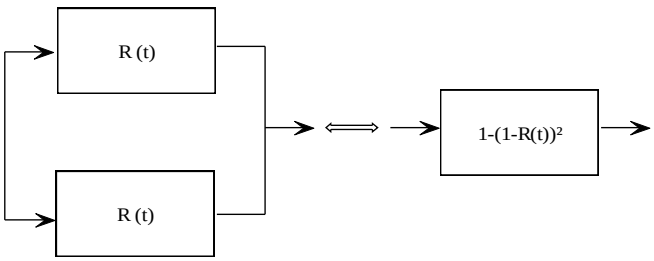
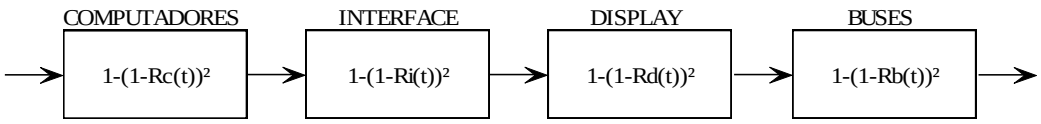


Figura 6.10

Seja Rc (t) a confiabilidade de um computador

Ri (t) a confiabilidade de uma interface,
Rd (t) a confiabilidade de um display e
Rb (t) a confiabilidade de um bus.

O diagrama de bloco resultante série é:



Rsistema = [1- (1- Rc (t)) ^{2}]. [1-(1-Ri (t) ^{2})]. [1-(1-Rd (t) ^{2})].[1-(1Rb (t)2) ^{2}]

Seja a título de exemplo a confiabilidade depois de 1hora.

RC (1) = Ri (1) = Rd (1) = Rb (1) = 0,9 Rsistema (1 hora) = 0,96.

Agora já se pode analisar a vantagem da redundância na confiabilidade.

O mesmo sistema anterior sem redundância tem uma confiabilidade depois de 1h de:

Rsistema (1 hora) = Rc (1h) .Ri (1h) .Rd (1h) .Rb (1h)

Rsistema (1h) = 0, 6561

6.8.1.3. Cobertura de Defeito e seu Impacto na Confiabilidade

Como definido anteriormente, cobertura de defeito é a medida da habilidade do sistema recuperar-se de defeitos.
Por exemplo, num sistema redundante antes que a redundância seja usada, requere-se uma boa cobertura de defeitos.
Durante a análise dos sistemas paralelos supõe-se que a cobertura dos defeitos era perfeita.
A cobertura não perfeita está no fato de o sistema não ser capaz de usar a redundância, pois não pode identificar que a unidade está em falha e, portanto, não a remove e não a troca por outra unidade livre de falha.

Para ilustrar veja a figura a seguir:

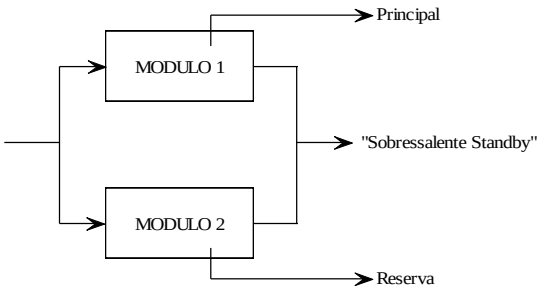


Figura 6.11

Este sistema paralelo de dois módulos funciona corretamente se uma das condições a seguir existir:

1. Módulo 1 está funcionando corretamente;
2. Módulo 2 está funcionando corretamente, módulo 1 falhou e a falha foi detectada e apropriadamente tratada.

A probabilidade que um destes eventos existe pode ser calculada como:

$$R_{sistema}(t) = R1(t) + (1 - R1(t)).C1R2(t)$$

onde C1 é a cobertura de defeitos associada ao módulo 1.

Se a confiabilidade e o fator de cobertura dos dois módulos são idênticos a expressão de confiabilidade é dada por:

$$R_{sistema}(t) = R(t) + R(t).C.(1-R(t))$$

Se o fator de cobertura for C=1, então:

$$R_{sistema}(t) = 2R(t) - R^2(t) = 1 - (1-R(t))^2$$

Se o fator de cobertura for C=0, então:

Rsistema (t) = R (t) que é a confiabilidade de **apenas um módulo.**

A figura a seguir, mostra o impacto do fator de cobertura na confiabilidade do sistema.

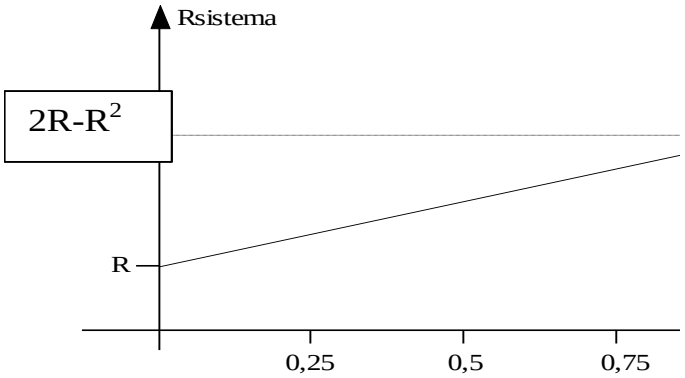


Figura 6.12

Neste modelo, o módulo 1 é o módulo principal enquanto estiver funcionando corretamente, e o sistema funciona corretamente, e o sistema funciona corretamente, até mesmo se o módulo 2 falha. Isto pode não ser verdade em sistemas que realizam comparações entre os dois módulos. Neste caso, se o módulo falho for detectado, o sistema continua operando com o módulo livre de defeito e o mecanismo de comparação é desabilitado. Se o módulo falho não for detectado, o sistema descontinua a operação. Também neste caso, deve ser considerada a cobertura de defeitos.

Vamos ilustra agora o exemplo anterior com comparação entre os dois módulos como meio de detecção de defeito.

Assume-se que a **comparação é perfeita** e detecta todas as falhas.

Uma vez que o processo de comparação detecte o defeito, o sistema implementa auto diagnóstico para determinar qual módulo está falho.

Se o defeito pode ser localizado, o módulo livre de defeito começa a desempenhar as funções do sistema.

O sistema funciona corretamente enquanto ambos os módulos estiverem funcionando corretamente ou um defeito ocorreu e foi detectado e manipulado (tratado) corretamente.

A confiabilidade do sistema pode ser colocada como:

$$R_{sistema}(t) = R1(t).R2(t) + \underbrace{R1(t).(1-R2(t)).C2}_{\text{módulo2 falhou}} + \underbrace{(1-R1(t)).C1.R2(t)}_{\text{módulo1 falhou}}$$

C1 - cobertura do auto - teste do módulo 1.

C2 - cobertura do auto - teste do módulo 2.

Se a confiabilidade e a cobertura de cada módulo são idênticas, então:

$R_{sistema}(t) = R^2(t) + 2R(t).C.(1-R(t))$

Para cobertura de defeito perfeita (C=1) têm-se:

$\underline{R_{sistema}(t) = R^2(t) + 2R(t)(1-R(t)) = 1-(1-R(t))^2}$

Se o fator de cobertura for zero (C= 0), então:

$R_{sistema}(t) = R^2(t)$

Caso o sistema deve parar de funcionar quando ambos os módulos discordarem então:

$R_{sistema}(t) = R1(t).R2(t)$ com funcionamento perfeito da **comparação.**

6.8.1.4. Sistema M de N

Estes sistemas são uma generalização do sistema paralelo ideal.

No sistema paralelo ideal, apenas **um** dos **N** módulos é requerido trabalhar para o sistema funcionar. Num sistema **M de N,** M módulos dos N devem funcionar para o sistema agir corretamente.

Um bom exemplo deste sistema é o TMR ("Redundância Modular Tripla") onde 2 módulos 3 devem agir no mecanismo de votação.

Vamos analisar um sistema TMR

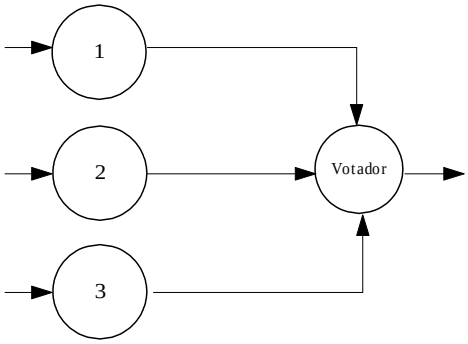


Figura 6.13

Obs.: Considerar que a Confiabilidade do Votador = 1.

$$RTMR(t) = R1(t).R2(t).R3(t) + R1(t).R2(t).(1-R3(t)) +$$

3 módulos funcionando — Módulo 3 falhou

$$+ R1(t) . (1-R2(t)) . R3(t) + (1-R1(t)) . R2(t) . R3(t)$$

módulo 2 falhou — módulo 1 falhou

Considerando $R1(t) = R2(t) = R3(t) \Rightarrow$

$$RTMR = R^3(t) + 3R^2(t) . (1-R(t)) =$$

$$= R^3(t) + 3R^2(t) - 3R^3 - (t)$$

$$\Rightarrow RTMR(t) = 3R^2(t) - 2R^3(t)$$

Vamos comparar a confiabilidade de RTMR com um módulo isolado.

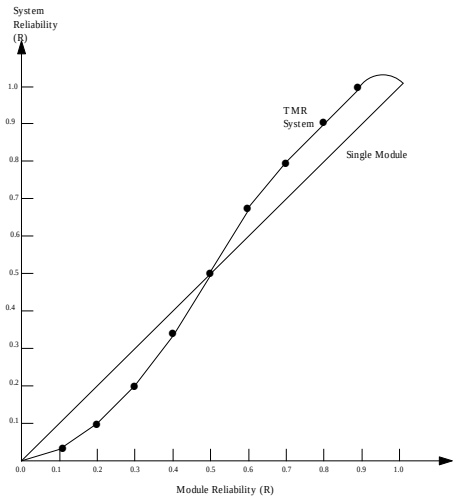


Figura 6.14

Esta figura mostra que existe um cruzamento entre a confiabilidade RTMR e R.

O ponto de cruzamento é:

$3R^2 - 3R^3 (t) = R \Rightarrow 3R - 2R^2 = 1$

{ R=0,5 e R=1}

$R^2 - \frac{3}{2} R + 0,5 = 0$

$\Delta = \frac{9}{4} - 2 = \frac{1}{4}$

$R = \frac{3/2 \pm ½}{2}$

Agora dá para ilustra melhor a diferença entre o conceito de tolerância a Defeito e Confiabilidade.

Um sistema pode ser Tolerante a Defeito e ainda ter uma baixa confiabilidade.

Um sistema TMR com módulos de confiabilidade R= 0,5 pode tolerar defeitos em um dos módulos, mas a sua confiabilidade é a mesma que a de um módulo único.

Um sistema com um módulo com alta confiabilidade não é tolerante a defeito.

Observação importante: É possível que a confiabilidade de um sistema não redundante se aproxime de um sistema redundante com os mesmos módulos, entretanto o sistema não redundante não será tolerante a defeito.

A expressão geral de um sistema M de N é dada por:

$$R_{M\text{-}N}(t) = \sum_{i=0}^{N-M} \binom{N}{i} R^{N-i}(t) \cdot (1-R(t))^{i}$$

.M funcionando

.i falhos

6.8.2 Modelos de Markov

Uma limitação do Modelo Combinatório é que não consegue modelar sistemas complexos. Pode ser bastante difícil construir o Diagrama de blocos de Confiabilidade.

O Modelo de Markov permite modelar o processo de reparo que ocorre em muitos sistemas, característica difícil de modelar em sistemas combinatórios.

Os dois principais conceitos no Modelo de Markov são os **Estados** e as **Transições.**

O **Estado** representa todas as situações conhecidas que descrevem o sistema num dado instante.

Para os modelos de confiabilidade, cada **estado** de Markov representa uma combinação diferente de módulos em falha e livres de falha.

Vamos analisar um sistema TMR. Definimos que S representa o estado do sistema, S = (S1, S2, S3) onde Si=1 se o módulo i está livre de falha e Si = 0 se o módulo i está falho.

São 8 estados possíveis.

As **transições** regulamentam as mudanças de estados que ocorrem dentro de um sistema.

O Diagrama de Estados do sistema TMR é apresentando a seguir:

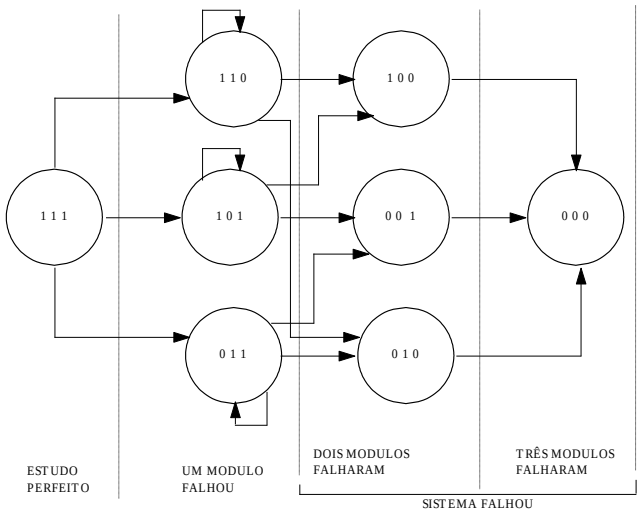


Figura 6.15

Vamos assumir que cada módulo no sistema TMR obedeça a lei de falha exponencial com uma taxa de falha λ, constante.

A probabilidade de um módulo estar falho no instante t é dado por $\Rightarrow$

$=> (1 - e^{-\lambda \Delta t})$

Da matemática quando Δt é pequeno, então:

$1 - e^{-\lambda \Delta t} \cong \lambda . \Delta t$

Pode-se agora, completar a figura anterior, colocando-se a probabilidade das transições:

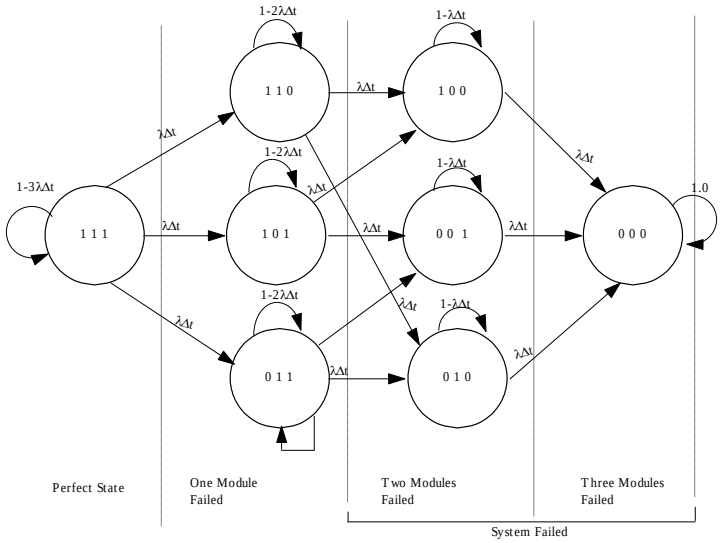


Figura 6.16

Reduzindo o Diagrama de Estados anterior, considerando os estados globais:

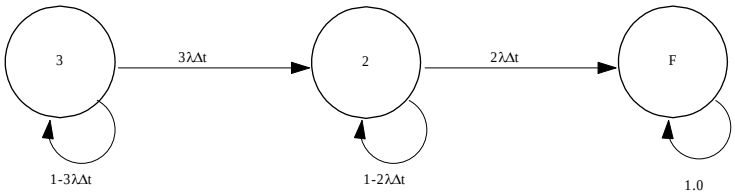
(Estão Perfeitos - Um módulo - Sistema falhou)

falhou

3 módulos 2 módulos ≤ 1 módulo

ok ok ok

- Diagrama de Estados Resultante é dado por:



$p3\ (t + \Delta t) = (1 - 3\lambda\Delta)\ .\ p3\ (t)$

↑

Probabilidade do sistema estar no estado 3 no instante t.

$p2\ (t+\Delta t) = (3\lambda\Delta t)p3\ (t) + 1 - 2\lambda\Delta t)\ .\ p2(t)$

$pF\ (t+\Delta t) = (2\lambda\Delta t)p2\ (t) + pF(t)$

As equações do Modelo de Markov para o sistema TMR podem ser escritas como:

$$\begin{bmatrix} p3\ (t+\Delta t) \\ p2\ (t+\Delta t) \\ pF\ (t+\Delta t) \end{bmatrix} = \begin{bmatrix} (1-3\lambda\Delta t) & 0 & 0 \\ 3\lambda\Delta t & (1-2\lambda\Delta t) & 0 \\ 0 & 2\lambda\Delta t & 1 \end{bmatrix} \begin{bmatrix} p3\ (t) \\ p2\ (t) \\ pF\ (t) \end{bmatrix}$$

$$P\ (t + \Delta t) = A.P\ (t)$$

$t=0 \qquad (P\ (\Delta t) = A.P\ (0))$

$t= \Delta t \qquad P\ (2\Delta t) = A.P\ (\Delta t) = A^2\ P\ (0)$

$$\Downarrow$$

$$P\ (n\Delta t) = A^n\ .P\ (0)$$

A probabilidade de um sistema falhar é dada pela probabilidade do sistema estar no estado falho.

$\Rightarrow$ RTMR (t) = 1 - PF (t) = P3 (t) + P2 (t)

Através de manipulação algébrica, tem-se:

$$\frac{p3\,(t+\Delta t) - [3\,(t)}{\Delta t} = -3\lambda.p3\,(t)$$

$$p2\,(t+\Delta t) = 3\lambda\Delta t\,.p3\,(t) + (1-2\lambda\Delta t\,(t))\,.\,p2\,(t)$$

$$p2\,(t+\Delta t) - p2(t) = 3\lambda\Delta t p3\,(t) - 2\lambda\Delta t\,(t)\,.\,p2\,(t)$$

$$\frac{pF\,(t+\Delta t) - pF(t)}{\Delta t} = 2.\lambda.p2\,(t)$$

Aplicando o limite $\Delta t \longrightarrow 0$:

$$\begin{cases} \dfrac{dp3\,(t)}{dt} = -3\lambda p3\,(t) \\ \dfrac{dp2\,(t)}{dt} = 3\lambda p3\,(t) - 2\lambda.p2\,(t) \\ \dfrac{dpF\,(t)}{dt} = 2.\lambda.p2\,(t) \end{cases}$$

Aplicando a **Transformação de Laplace:**

$$sP3\,(s) - p3\,(0) = -3\lambda P3\,(s)$$

$$sP2\,(s) - p2\,(0) = 3\lambda P3\,(s) - 2\lambda P2(s)$$

$$sPF(s) - pF\,(0) = 2\lambda P2\,(s)$$

Como no instante inicial (t=0) estamos supondo que o sistema esteja perfeito

$\Rightarrow$

$$p3\,(0) = 1,\ p2\,(0) = 0,\ \text{e}\ pF\,(0) = 0$$

$$P3(s) = \frac{1}{s+3\lambda}$$

$$P2(s) = \frac{3\lambda}{(s+2\lambda)(s+3\lambda)} = \frac{3}{(s+2\lambda)} + \frac{-3}{(s+3\lambda)}$$

$$PF(s) = \frac{6\lambda^2}{(s+2\lambda)(s+3\lambda)} = \frac{1}{s} + \frac{-3}{(s+2\lambda)} + \frac{2}{(s+3\lambda)}$$

$$p3(t) = e^{-3\lambda t}$$

$$p2(t) = 3e^{-2\lambda t} - 3e^{-3\lambda t}$$

$$pF(t) = 1 - 3.e^{-2\lambda t} + 2e^{-3\lambda t}$$

Desta forma a confiabilidade do sistema:

$$RTMR(t) = p3(t) + p2(t) = e^{-3\lambda t} + 3e^{-2\lambda t} - 3e^{-3\lambda t}$$

$$= 3e^{-2\lambda t} - 2e^{-3\lambda t}$$

	Reliability	
Time (t) in minutes	Combinatorial results	Markov results
1	0.99999177	0.99999171
2	0.99996674	0.99996686
3	0.99992549	0.99992561
4	0.99986792	0.99986809
5	0.99979424	0.99979442
6	0.99970472	0.99970472
7	0.99959898	0.99959916
8	0.99947786	0.99947786
9	0.99934101	0.99934095
10	0.99918842	0.99948854

Failure rate λ is 0.1 failures per hour, and time step Δt is 0.1 seconds

6.8.2.1.Modelo de Markov com Cobertura de Defeitos

Analisamos o Modelo de Markov, quando usando em sistemas que não dependem de cobertura de defeitos ou processo de reparo.

Veremos agora o Modelo de Markov em sistemas que dependem da cobertura dos defeitos.

O sistema a ser modelado é de redundância tripla que usa técnicas de deteção de defeitos para detectar a ocorrência de um defeito em um dos três módulos independentes.

O modo correto de um módulo defeituoso não interferir é ser removido através da abertura de uma chave.

A probabilidade que um defeito seja corretamente manipulado corresponde à cobertura dos defeitos, denotada por C.

A arquitetura básica do sistema é mostrada na figura que segue:

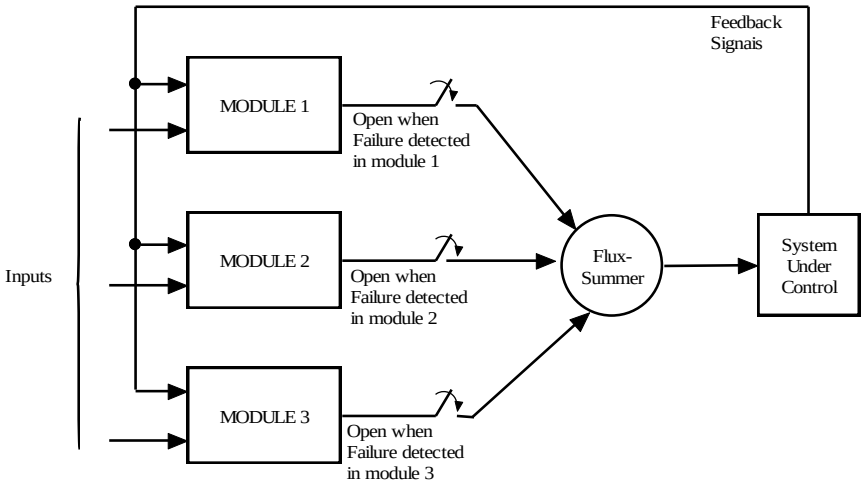


Figura 6.17

O Modelo de Markov está apresentado a seguir. Existem 4 erros neste modelo! Ache-os!!

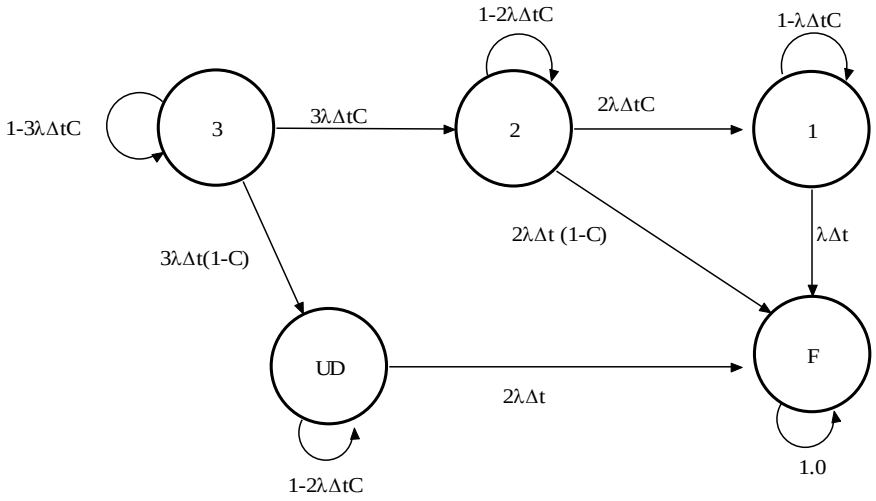


Figura 6.18

O sistema é assumido começar num estado sem defeito (estado 3).

Há dois caminhos a partir deste estado. O primeiro é a transição para o estado 2 e corresponde a um defeito em um dos três módulos detectável (dentro da cobertura).

O segundo é uma transição para o estão UD que corresponde a um dos três módulos estarem com defeito sem ser detectável.

Quando o sistema entra no estado UD torna-se um sistema TMR básico com votação majoritária.
Quando o sistema está no estado 2 pode tolerar um defeito detectável para o estado 1, ou caso seja um defeito não detectável o sistema vai para o estado falho.
As equações para o Modelo de Markov são:

$$\begin{cases} p3(t+\Delta t) = p3(t).(1-3\lambda\Delta t) \\ p2(t+\Delta t) = p3(t).3.\lambda.\Delta t.\ C + p1(1-2\lambda\Delta t) \\ p1(t+\Delta t) = p2(t).2.\ \lambda.\Delta t.\ C + p1(t).(1-\lambda\Delta t) \\ puD(t+\Delta t) = p3(t).3.\ \lambda.\Delta t\ (1-C) + puD(t).(1-2\lambda\Delta t) \\ pF(t+\Delta t) = p2(t).2\ \lambda.\Delta t\ (1-C) + p1(t)\ \lambda.\Delta t + puD(t).2\lambda\Delta t + pF(t) \end{cases}$$

A confiabilidade do sistema é a probabilidade de estar nos estados 3, 2, 1, ou UD.

$$R(t) = p3(t) + p2(t) + p1(t) + pUD(t).$$

A tabela a seguir mostra a confiabilidade em função da cobertura.

Fault coverage	Reliability (after 1 hour)
0.0	0.97460
0.1	0.97484
0.2	0.97558
0.3	0.97680
0.4	0.97852
0.5	0.98073
0.6	0.98343
0.7	0.98662
0.8	0.99030
0.9	0.99448
1.0	0.99914

$\frac{\Delta R}{\Delta C} = 0{,}0024$

$\frac{\Delta R}{\Delta C} = 0{,}0418$

Failure rate λ is 0.1 failures per hour, and time step Δt is 0.1 seconds

$$\begin{pmatrix} p_3(t+\Delta t) \\ p_2(t+\Delta t) \\ p_1(t+\Delta t) \\ p_{UD}(t+\Delta t) \\ p_F(t+\Delta t) \end{pmatrix} = \begin{pmatrix} 1-3\lambda\Delta t & 0 & 0 & 0 \\ 3\lambda\Delta tC & 1-2\lambda\Delta t & 0 & 0 \\ 0 & 2\lambda\Delta tC & 1-\lambda\Delta t & 0 \\ 3\lambda\Delta t(1-C) & 0 & 0 & 1-2\lambda\Delta t \\ 0 & 2\lambda\Delta t(1-C) & \lambda\Delta t & 2\lambda\Delta t \end{pmatrix} x \begin{pmatrix} p_3(t) \\ p_2(t) \\ p_1(t) \\ p_{LD}(t) \\ p_F(t) \end{pmatrix}$$

Quando o fator de cobertura é zero, o sistema é idêntico a um sistema TMR com votação majoritária.

A tabela mostra que há um impacto maior na confiabilidade em valores de maior cobertura.

6.8.2.2. Modelo de Markov com Reparo

Considera-se agora, o sistema com reparo como forma de recuperação. Considere o Modelo de Markov de um sistema simples consistindo de um computador sem redundância e assuma que o computador tenha uma taxa de falha constante de λ e taxa de reparo de μ**.**

O Modelo de Markov deste sistema é apresentado a seguir:

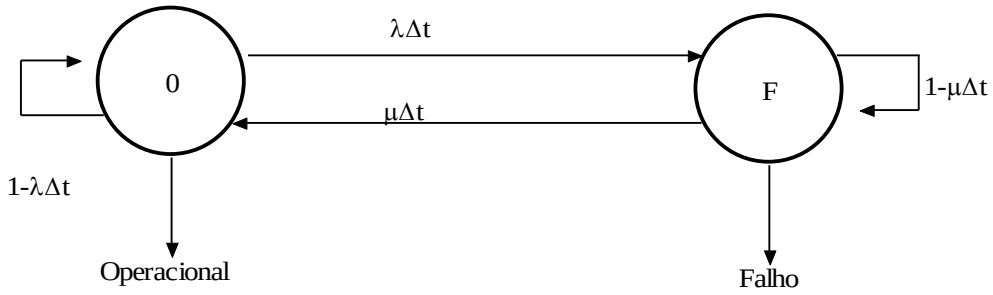


Figura 6.19

As equações deste modelo são:

$$\begin{cases} po\,(t+\Delta t) = po(t)\,(1-\lambda\Delta t)+pF(t)\mu\Delta t \\ pF\,(t+\Delta t) = po(t)\,\lambda\Delta t)+pF(t)\,(1-\mu\Delta t) \end{cases}$$

$$\begin{cases} \dfrac{po\,(t+\Delta t) - po(t)}{\Delta t} = -\lambda po(t)+\mu pF(t) \\ \dfrac{pF\,(t+\Delta t) - pF(t)}{\Delta t} = -\lambda po(t)+\mu pF(t) \end{cases}$$

$\Delta t \rightarrow 0$

$$\begin{cases} \dfrac{dpo\,(t)}{\Delta t} = -\lambda po(t)+\mu pF(t) \\ \dfrac{dpo\,(t)}{\Delta t} = -\lambda po(t) -\mu pF(t) \end{cases}$$

Condições iniciais: po (0) = 1 pF (0) = 0

Passando para o domínio S = (Laplace)

$$\begin{cases} Spo(S) - po(0) = -\lambda Po(S) + \mu PF(S) \\ SPF(S) - pF(0) = -\lambda Po(S) + \mu Pi(S) \end{cases}$$

$$\begin{cases} SPo(S) = 1 - \lambda Po(S) + \mu PF(S) \\ SPF(S) = \lambda Po(S) - \mu PF(S) \end{cases}$$

$$\begin{cases} Po(s) = \dfrac{1}{S+(\lambda+\mu)} + \dfrac{\mu}{S+(\lambda+\mu)} \\ Po(s) = \dfrac{\lambda}{S(S+(\lambda+\mu))} \end{cases}$$

$$\begin{cases} Po(s) = \dfrac{\frac{\mu}{\lambda+\mu}}{S} + \dfrac{\frac{\lambda}{\lambda+\mu}}{S+(\lambda+\mu)} \\ Po(s) = \dfrac{\frac{\lambda}{\lambda+\mu}}{S} + \dfrac{\frac{\lambda}{\lambda+\mu}}{S+(\lambda+\mu)} \end{cases}$$

Aplicando a transformada de Laplace inversa:

$$\begin{cases} Po(s) = \dfrac{\mu}{\lambda+\mu} + \dfrac{\lambda}{\lambda+\mu} \cdot e^{-(\lambda+\mu)t} \\ Po(s) = \dfrac{\lambda}{\lambda+\mu} + \dfrac{\lambda}{\lambda+\mu} \cdot e^{-(\lambda+\mu)t} \end{cases}$$

Para t=0 → po (0) = 1 e pF (0) = 0

Para $t \rightarrow \infty \Rightarrow$

$$po(\infty) = \frac{\mu}{\lambda+\mu} = \frac{1}{\frac{\lambda}{\mu} + 1}$$

Se $\mu >> \lambda \rightarrow \frac{\lambda}{\mu} = 0 \rightarrow po(\infty) = 1$

Se $\lambda >> \mu \rightarrow \frac{\lambda}{\mu} = \infty \rightarrow po(\infty) = \phi$

$$pF(\infty) = \frac{\lambda}{\lambda+\mu} = \frac{1}{\frac{1+\mu}{\lambda}}$$

A seguir é apresentado um gráfico que mostra a probabilidade de um sistema estar operacional (Po(t) variando a taxa de reparo (μ) e considerando a taxa de falha λ - 0,1 falhas/h) Δt= 0,1 segundos.

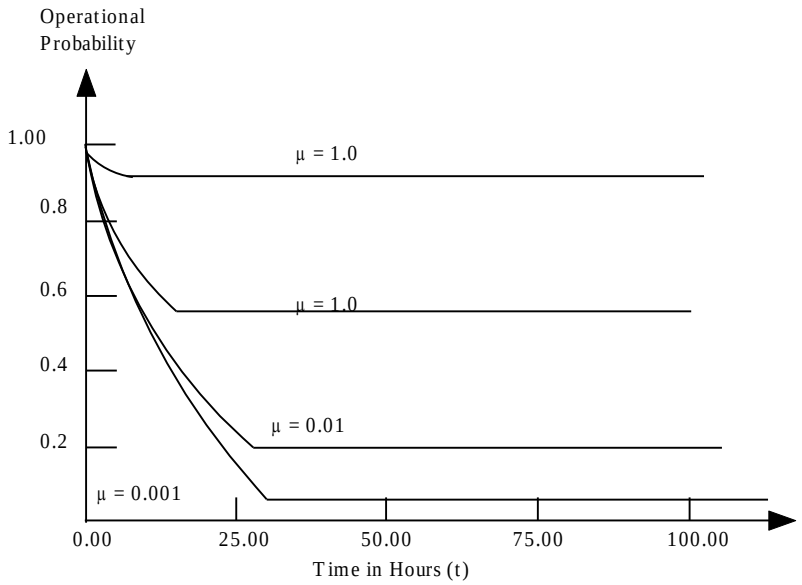


Figura 6.20

6.8.2.3. Modelo de Segurança

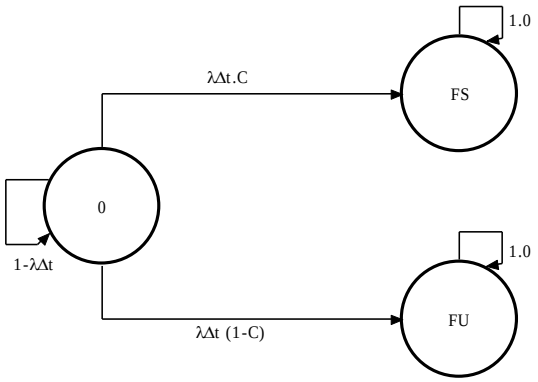
A definição de Segurança diz ser a probabilidade do sistema ou funcionar corretamente ou falhar de maneira segura.

As definições de estados Seguros e Inseguros devem ser criadas para cada aplicação.

No modelo de Markov o estado falho deve ser, explodido em FS, falha segura, e FU, falha insegura.

O modelo de Markov para um sistema com um único módulo com taxa de falha e auto diagnóstico com cobertura C é mostrado a seguir. Neste sistema as falhas seguras são consideradas aquelas detectadas pelo auto - diagnóstico.

Figura 6.21



A segurança do sistema é definida como:

$$S(t) = po(t) + pFS(t)$$

As equações do modelo são:

$$\begin{cases} po(t+\Delta t) = (1-\lambda\Delta t) \cdot po(t) \\ pFS(t+\Delta t) = \lambda.\Delta t.C.po(t)+pFS(t) \\ pFU(t+\Delta t) = \lambda.\Delta t.(1-C)po(t)+pFU(t) \end{cases}$$

$t \rightarrow 0$

$$\begin{cases} \frac{dpo(t)}{t} = \lambda po(t) \\ \frac{dpFS(t)}{dt} = \lambda.C.po(t) \\ \frac{dpFU(t)}{dt} = \lambda (1-C).po(t) \end{cases}$$

Aplicando a transformada Laplace:

$$\begin{cases} Po(S) = \frac{po(0)}{S+\lambda} \\ PFS(S) = \frac{\lambda.C.Po(0)}{S(S+\lambda)} + \frac{PFS(0)}{S} \\ PFU(S) = \frac{\lambda.(1-C)Po(0)}{S(S+\lambda)} + \frac{PFU(0)}{S} \end{cases}$$

onde:

$po(0) = 1$, $pFS(0) = pFU(0) = 0$

$$Po(S) = \frac{1}{S+\lambda}$$

$$PFS(S) = \frac{\lambda.C.}{S(S+\lambda)} = \frac{(1-C)}{S} - \frac{(1-C)}{S+\lambda}$$

$$PFU(S) = \frac{\lambda.(1-C)}{S(S+\lambda)} = \frac{(1-C)}{S} - \frac{(1-C)}{S+\lambda}$$

Anti-transformando tem-se:

$Po(t) = e^{-\lambda t}$

$pFS(t) = C - C.e^{-\lambda t}$

.

$pFS(t) = (1-C) - (1-C).e^{-\lambda t}$

A confiabilidade do sistema é:

$R(t) = Po(t) = e^{-\lambda t}$

A segurança do sistema é dada por:

$S(t) = po(t) + p_{FS}(t) = C + (1-C)\, e^{-\lambda t}$

No instante t=0 a Segurança é 1.

$S(\infty) = C$

A Segurança de um sistema está diretamente dependente da cobertura de detecção de defeito.

6.8.2.4 Comparação entre Sistemas

Pretende-se comparar o MTTF de dois sistemas, ou sua confiabilidade.
Os sistemas a serem comparados são o Simples e o TMR.
Assume-se que a votação é perfeita.

O $MTTF = \int_0^\infty R(t)\, dt$, onde

R(t) é a confiabilidade do sistema.

$$R_{simples} = \int_0^\infty e^{-\lambda t}.dt = \frac{1}{\lambda}$$

$$MTTF_{TMR} = \int_0^\infty (3.e^{-2\lambda t} - 2.e^{-3\lambda t}).dt = \frac{5}{6\lambda}$$

O gráfico que segue mostra a confiabilidade em função de λt.

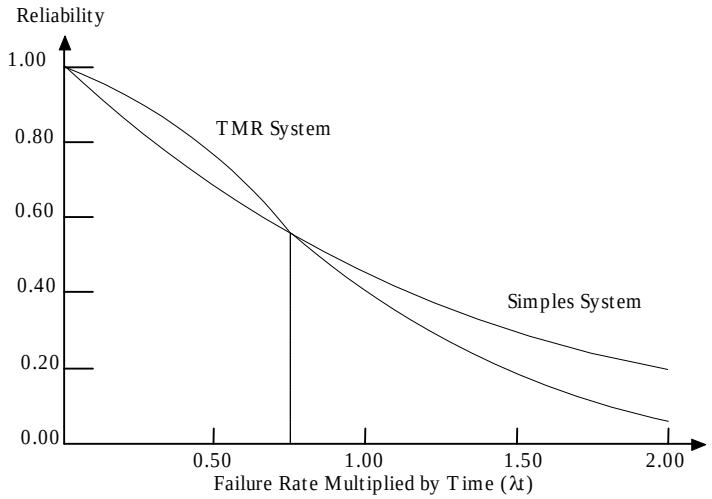


Figura 6.22

Para certos valores λt, a confiabilidade do sistema TMR é inferior em relação ao Sistema simples.

Desta forma, o MTTF pode não representar adequadamente a qualidade do sistema.

Em alguns casos é mais importante determinar o TEMPO DA MISSÃO, que é o tempo requerido no qual a confiabilidade do sistema permaneça superior a um determinado nível.

Seja o sistema simples e o nível de confiabilidade de missão r:

$$r = e^{-\lambda t}$$

$$\ln r = -\lambda . t$$

$$t = \frac{-1nr}{\lambda}$$

Tempo da Missão

Para um sistema TMR:

$$r = 3\, e^{-2\lambda t} - 2.e^{-3\lambda t}$$

Para um r = 0,86 e λ=0,01 falhas/h

$T_{MISSÃO\ -}$ simples = 15,08 horas e

$T_{MISSÃO\ -}$ TMR = 27 horas e

Em outras palavras, um sistema TMR pode operar, neste exemplo, 1,8 vezes mais tempo mantendo uma confiabilidade maior do que 0,86.

Seja um exemplo ilustrativo apresentado a seguir.

Apesar da confiabilidade do sistema (1) tender a zero quando λt cresce, o tempo de missão t1 é maior do que o tempo de missão t2 para a curva (2).

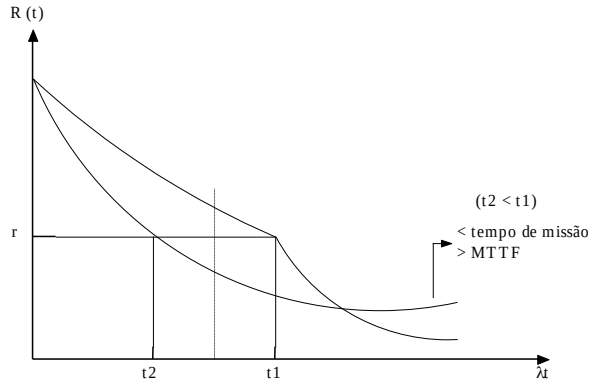


Figura 6.23

6.8.2.5. Modelo de Disponibilidade.

A taxa de reparo pode afetar drasticamente a disponibilidade de um sistema.

A disponibilidade de um sistema A (t), é definida como a probabilidade do sistema estar disponível para desempenhar suas tarefas no instante de tempo t.

Intuitivamente, a disponibilidade pode ser aproximada como o tempo total que um sistema está operacional dividido pelo tempo total que o sistema foi colocado em operação. Em outras palavras, a disponibilidade é a porcentagem do tempo que o sistema está disponível para realizar as operações esperadas.

$$A\ (t_{corrente}) = \frac{\text{toperacional}}{\text{toperacional} + \text{treparo}}$$

A avaliação experimental é muito difícil devido ao tempo e custo envolvido.

Deve-se então, considerar duas abordagens. A primeira é baseada em parâmetros como MTTF e MTTR e define a chamada Disponibilidade Estável ("Steady - state availabitity" - ASS).

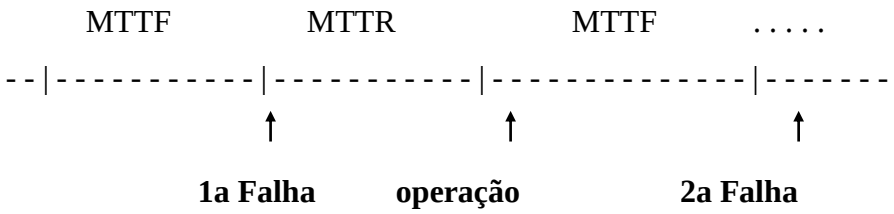


Figura 6.24

$$ASS = \frac{N(MTTF)}{N(MTTF) + N(MTTR)} = \frac{MTTF}{MTTF + MTTR}$$

$$MTTF = \frac{1}{\lambda}, \quad MTTR = \frac{1}{\mu} \text{ (P/sistema simples)}$$

$$Ass = \frac{\frac{1}{\lambda}}{\frac{1}{\lambda} + \frac{1}{\mu}}$$

Exemplo : λ = 0,01 falhas/ hora
μ = 0,1 reparos/ hora
} Ass = 0.90909

A outra abordagem usa a taxa de falhas e a taxa de reparo no Modelo de Markov para calcular a disponibilidade em função do tempo.

Considere agora o diagrama de Markov para um sistema simples, com reparo:

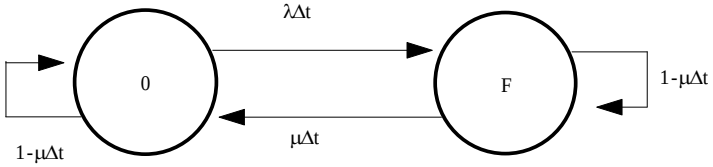


Figura 6.25

$po(t+\Delta t) = po(t)(1-\lambda\Delta t) + pF(t).\ \mu\ \Delta t$

$pF(t+\Delta t) = po(t)\ \lambda\ \Delta t + pF(t).(1-\mu\Delta t)$

Po (t) probabilidade de o sistema estar operacional no instante t (Disponibilidade)

$$po(t) = \frac{\mu}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu} \cdot e^{-(\lambda+\mu)t}$$

Quando $t \rightarrow \infty \rightarrow po(\infty) = \frac{\mu}{\lambda+\mu} = \frac{1}{1+\frac{\lambda}{\mu}}$ = ASS

Para λ = 0,01 falhas/h e μ=0,01 reparos/h o gráfico de disponibilidade está apresentado a seguir.

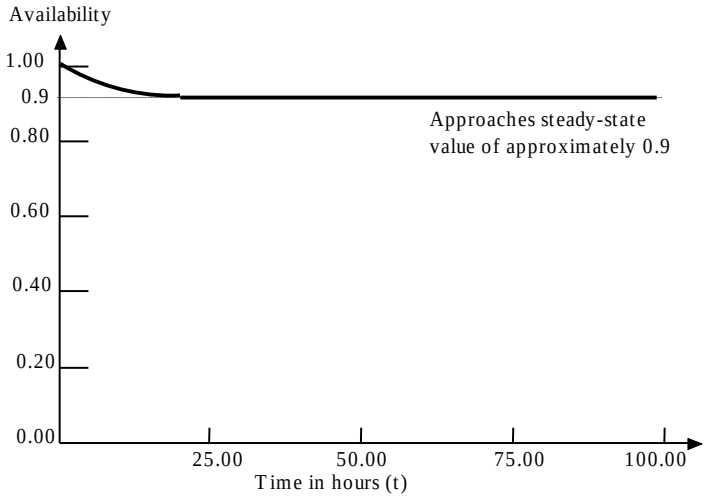


Figura 6.26

6.8.2.6. Modelo de Manutenabilidade

A Manutenabilidade M (t) é a probabilidade de um sistema falho ser reparado dentro de um tempo especificado t.

Um parâmetro importante aqui é a taxa de reparo**.** A taxa de reparo é o número médio de reparos que pode ser realizado por unidade de tempo.

O inverso da taxa de reparo é o MTTR, que é o tempo médio requerido para realizar um único reparo.

$$MTTR = \frac{1}{\mu}$$

Um sistema pode ser construído e defeitos injetados. O tempo médio requerido para reparar o sistema é medido e anotado como MTTR. Uma boa estimativa do MTTR pode ser obtida apenas se um número suficiente de defeitos diferentes é injetado, e grupo de reparo, com níveis variados de profissionais.

Suponha que se tenham N sistemas. Injeta-se um único defeito em cada sistema e uma pessoa da manutenção vai reparar cada sistema. Os defeitos são injetados no instante t = 0.

No instante t, determina-se Nr(t) o número de sistema reparados e Nnr(t) o número de sistemas não reparados.

$$M(t) = \frac{Nr(t)}{N} = \frac{Nr(t)}{Nr(t) + Nnr(t)}$$

$$\frac{dM(t)}{dt} = \frac{1}{N}\frac{dNr(t)}{dt}$$

$$\frac{dNr(t)}{dt} = N \cdot \frac{dM(t)}{dt}$$

Define-se $\frac{1}{Nnr(t)} \cdot \frac{dNr(t)}{dt} = \mu$ é a função taxa de reparo

$$\mu = \frac{1}{Nnr(t)} \cdot \frac{dNr(t)}{dt} = \frac{N}{Nnr(t)} \cdot \frac{dM(t)}{dt}$$

$$\frac{dM(t)}{dt} = \mu \cdot \frac{Nnr(t)}{N}$$

$$\frac{Nr(t)}{N} = M(t) = \frac{Nnr(t)}{N}$$

$$M(t) = 1 - \frac{Nnr(t)}{N} \rightarrow \frac{Nnr(t)}{N} = 1 - M(t)$$

$$\frac{dM(t)}{dt} = \mu\,(1 - M(t))$$

$$\rightarrow M(t) = 1 - e^{-\mu t}$$

Se $\mu = 0 \rightarrow M(t) = 0$ (O sistema não pode ser reparado)

Se $\mu = \infty \rightarrow M(t) = 1$ (A manutenção pode ser realizada no tempo zero)

Quando t = MTTR

$$M(MTTR) = 1 - e^{(-\mu./\mu)} = 1 - e^{-1}$$

M (MTTR) = 0,632, ou seja,

O sistema apresenta a probabilidade de 0,632 de ser reparado num tempo menor ou igual ao MTTR.

Como a manutenção pode diferenciar muito em função do tipo de falha, a taxa de reparo é tipicamente especificada por nível de reparo.

A Divisão mais comum apresenta três níveis de reparo:

- Nível organizacional (menos de 1hora) : todos reparos que podem ser realizados no local do sistema;

Ex: troca de cartões;

- Nível intermediário (algumas horas) : todos reparos são realizados próximos ao sistema, num laboratório;
- Nível de fabrica (alguns dias): o equipamento deve ser reparado na fábrica.

Exemplo: Para um sistema de computador, tem-se 2 horas de reparo no nível organizacional, 8 horas no nível intermediário e 168 horas (1 semana) no nível de fábrica.

O gráfico a seguir apresenta a diferença entre a Manutenabilidade em cada caso.

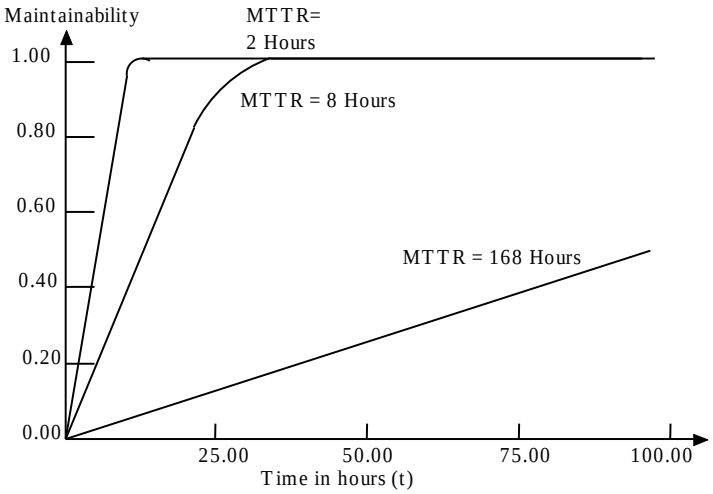


Figura 6.27

7. Segurança

Este capítulo tem como objetivo a apresentação dos principais conceitos na área de segurança, as causas fundamentais dos acidentes, envolvendo aspectos gerenciais, técnicos e humanos, além da importância da cultura de segurança como uma atividade presente em todo ciclo de vida de um sistema crítico. É apresentada também uma sugestão de padronização de terminologia discutida e utilizada dentro do GAS.

7.1. Aspectos Conceituais

O conceito de segurança, dentro deste trabalho, pode ser definido como a probabilidade de um sistema desempenhar, num determinado período de tempo, suas funções ou descontinuá-las sem causar mortes, danos à saúde, destruição de propriedades, perda de missão ou danos ao meio ambiente. Neste sentido, o termo segurança, neste trabalho, corresponde ao termo inglês "safety".

Para que se possa garantir a segurança de um sistema, devem ser utilizadas técnicas que permitam prevenir os acidentes previsíveis, bem como minimizar as conseqüências daqueles imprevisíveis. Em um acidente, são consideradas as perdas em geral, tais como destruição de propriedade, cancelamento de missões ou danos causados ao ambiente, além de ferimentos ou morte causados em seres humanos. O desenvolvimento desses sistemas considerados críticos é, normalmente, controlado por regulamentação governamental, que estabelece critérios de certificação de sistemas em cada área de aplicação.

Quando uma perda é considerada como grave ou de vulto, geralmente justifica os esforços e os recursos a serem investidos para sua prevenção. O valor a ser investido é considerado válido ou justificável, considerando-se tanto os aspectos técnicos, quanto outros fatores, tais como sociais, psicológicos, políticos e econômicos.

As atividades relativas à segurança devem se iniciar quando o sistema começa a ser concebido e tiver seqüência no projeto, produção, teste e operação do sistema. A segurança deve ser considerada como um todo, ou seja, não é suficiente que se assegure apenas a correção de partes ou de sub-sistemas de um sistema maior.

Desta forma, é de fundamental importância ter-se uma atividade de garantia da segurança atuante em todas as fases do ciclo de vida de um sistema, bem como os problemas, apontados durante a análise de risco, devem ser seriamente considerados, nunca se desprezando qualquer indício ou suspeita que possa vir a provocar um acidente. Daí a importância em se adotar e desenvolver metodologias e métodos que cada vez mais assegurem a qualidade dos trabalhos desenvolvidos na garantia da segurança de sistemas, de forma que se obtenham sistemas de funcionamento robusto.

No contexto dos sistemas críticos quanto à segurança, os estados inseguros correspondem àqueles estados também denominados perigosos, onde o sistema está exposto à ocorrência de um acidente. Neste trabalho é adotado o termo "estado perigoso". Um projeto que tenha de contemplar aspectos referentes à segurança de um sistema deve, em primeiro lugar, buscar a eliminação de estados perigosos. Se isto não for possível, deve ser alcançado o controle desses estados perigosos preferencialmente por meio de dispositivos passivos, baseados em processos físicos, tais como a força da gravidade. Novamente, se não houver possibilidade desse controle, deve ser buscada a redução de eventuais danos causados pela ocorrência do estado perigoso. A segurança de um sistema deve ser eficaz, evitando ou minimizando a ocorrência de acidentes, além de ter um custo compatível com o sistema considerado como um todo. Sua validade só é comprovada se acidentes forem realmente prevenidos ou evitados.

Algumas vezes, a segurança de um sistema atua como uma restrição aos projetos, pois alguns de seus requisitos podem entrar em conflito com aspectos operacionais e de desempenho, bem como podem ocasionar aumentos nos custos envolvidos.

Um acidente pode acontecer se houver alguma falha ou entrada imprópria não prevista ou não coberta pelos dispositivos que deveriam garantir a segurança de um sistema. Outra causa da ocorrência de acidentes deve-se ao fator humano, ou seja, uma falha de operação por parte de um ser humano, que ocasione uma condição que possa levar a um acidente, condição esta não prevista ou não coberta pelo projeto e pela implementação do sistema.

Desta forma, as causas que justificam a ocorrência de um acidente podem ter origem em deficiências na cultura de segurança das instituições, em falhas na estrutura organizacional dessas mesmas instituições ou ainda em atividades técnicas superficiais ou ineficientes, relativas à segurança.

Cada uma dessas formas é a seguir discutida.

a) Aspectos relacionados com as deficiências na cultura de segurança das organizações.

Requisitos desejáveis em um sistema podem ser conflitantes entre si, sendo necessário, portanto, estabelecer compromissos para o desenvolvimento do projeto. Neste aspecto, pode ocorrer que a melhor ou mais avançada tecnologia seja invalidada por decisões incorretas. Pode-se citar, como exemplo, o conflito entre os fatores disponibilidade e segurança. Muitas vezes as pressões existentes podem forçar o comprometimento da segurança em função da obtenção de maiores níveis de disponibilidade. Outro campo de pesquisa e discussão que vem ganhando destaque refere-se ao confrontos entre os conceitos de "Safety" e "Security", ambos denominados "segurança" na língua portuguesa. Com a tendência crescente da utilização de sistemas integrados via redes de computadores em aplicações críticas de segurança, pode-se haver conflitos entre requisitos de "security" e requisitos de "safety". Se os requisitos de "safety" e "security" forem definidos isoladamente, há o perigo que incongruências não reconhecidas e discutidas, e portanto não resolvidas, possam ocorrer, comprometendo a segurança final do sistema. Talvez a forma de integração mais apropriada seja a harmonização entre os processos de verificação de cada um desses aspectos, permitindo que os conflitos entre ambos sejam reconhecidos e resolvidos antecipadamente. [Eames 99]

Outro aspecto relacionado com a cultura de segurança refere-se à complacência e autoconfiança. As pessoas normalmente desenvolvem uma mentalidade sobre a infalibilidade do equipamento a que estão acostumadas a lidar, fruto de repetidas afirmações sobre a garantia da tecnologia utilizada para aumentar a segurança do sistema. Acredita-se que um acidente não possa ocorrer, pois não seria possível que houvesse a ocorrência de tantas condições adversas simultaneamente, o que nem sempre é verdade. Os piores acidentes ocorrem quando não se espera que possam vir a acontecer, pois geralmente se

gera uma acomodação por parte das pessoas. Ao contrário, quando se acredita na possibilidade de ocorrência de um acidente, são tomadas providências para preveni-lo ou minimizar seus efeitos.

As técnicas de redundância e diversidade de projetos, utilizadas em alguns sistemas para prevenir acidentes e aumentar a segurança, também não devem ser encaradas como a solução ótima para todos os casos. Muitas vezes, há as falhas de modo comum, que podem afetar todos os canais ou parâmetros ao mesmo tempo.

Normalmente as condições perigosas mais evidentes são as que recebem maior atenção e são, por conseguinte, controladas, enquanto que aquelas condições com menor probabilidade de ocorrência são desprezadas. É comum se verificar que, após a ocorrência de um acidente, sua causa era um evento conhecido, que foi desprezado, considerando sua ocorrência como improvável.

Outro fator importante a ser considerado no que diz respeito à complacência assumida é o caso em que um sistema opera por longos períodos de tempo sem falhas. Neste caso, acredita-se que o sistema não irá falhar nunca mais, o que não é verdade. De certo modo, o risco da ocorrência de um acidente pode até aumentar, devido a alterações no ambiente em que o sistema estiver inserido.

Também não podem ser ignorados sinais de alarme, pois os acidentes são freqüentemente precedidos por alertas, ou por uma série de ocorrências menores, geralmente ignoradas, pois não se acredita que algo mais grave ainda possa vir a acontecer.

b) Aspectos relacionados com problemas na estrutura organizacional das instituições.

A busca pela segurança deve vir desde os mais altos escalões de uma organização, difundindo-se em todos os setores da instituição. Agências governamentais e grupos de usuários podem tentar fazer com que a segurança seja mais seriamente considerada, o que só se tornará mais eficaz se houver uma conscientização, por parte da sociedade em geral, a respeito da importância fundamental das atividades que garantam a segurança dos sistemas considerados mais críticos.

De particular interesse são os sistemas compostos por diversos sub-sistemas, onde é necessário que haja um órgão centralizador de todos os grupos

envolvidos no desenvolvimento do sistema, de forma que se atribua qual, ou quais grupos devem se preocupar com atributos de segurança, principalmente nas interfaces entre os sub-sistemas.

Outro ponto a ser considerado com relação ao grupo responsável pela segurança, é que deve ser independente, no que diz respeito às equipes de projeto. De preferência é recomendável que tal grupo seja vinculado a entidades completamente independentes daquelas que estiverem realizando o projeto.

Devem existir canais de comunicação adequados, entre os diversos grupos envolvidos com o sistema, de forma tal que as metas desejáveis possam ser transmitidas dos níveis hierárquicos mais altos para os mais baixos, e no sentido contrário, permitindo a avaliação sobre a evolução do projeto.

c) Problemas decorridos de atividades técnicas superficiais ou ineficientes quanto à segurança.

As condições perigosas devem ser cuidadosamente analisadas, com registros que justifiquem e suportem cada decisão de projeto, bem como os compromissos assumidos entre demais fatores e segurança.

Muitas vezes, esforços no sentido de garantir a segurança não são eficazes, pois são eliminadas as causas específicas de acidentes, mas não as causas básicas. Outro ponto que ocorre é que o projeto pode ser baseado em falsas suposições, como por exemplo, independência entre os eventos. Pode acontecer ainda de que eventuais modificações para aumentar a segurança acabem por ocasionar efeito contrário, ou seja, incrementar o número de estados perigosos devido ao aumento da complexidade do sistema. Pode ocorrer que a colocação de um dispositivo de segurança instalado para corrigir determinada condição, seja utilizada para justificar redução em margens de segurança em outros aspectos.

Outro fator muito importante é a realimentação com as informações sobre os acidentes ocorridos em outros sistemas, similares ou não, ao que estiver sendo desenvolvido, pois essas informações podem e devem ser efetivamente utilizadas na prevenção de novos acidentes.

7.2. Erro Humano

Muitos fatores podem levar o ser humano a agir de forma incorreta nos sistemas críticos quanto à segurança. Neste aspecto está sendo feita referência especial aos Operadores. Pode ser citado, como exemplo, o fornecimento ao operador de dados incompletos, incorretos, complexos ou excessivos. Outra crença negativa é que os operadores podem superar qualquer emergência, forçando-os a intervir em situações limites.

Além destes aspectos, muitas vezes o operador é responsabilizado por acidentes como forma de negligenciar ou até mesmo esconder erros cometidos por projetistas e gerentes. Na grande maioria das vezes, as ações positivas dos operadores raramente são destacadas, sendo registradas apenas as ações negativas.

Tendo em mente todas essas dificuldades, é apresentado a seguir uma breve discussão sobre a necessidade dos operadores nos sistemas automáticos, os modelos cognitivos da tarefa humana e os possíveis papéis dos operadores nestes sistemas.[Leveson 95]

a) A Necessidade do Operador em Sistemas Críticos quanto à Segurança

Uma das grandes questões que surge se refere à eliminação ou não do operador dos sistemas críticos, substituindo-os por computadores. Diversas razões corroboram para que o operador não seja eliminado desses sistemas. São discutidos a seguir alguns destes aspectos.

Pode-se afirmar que é extremamente difícil antecipar todas as condições e todas as interações não desejadas entre os componentes, que podem ocorrer no ambiente de um sistema. A presença do operador nestes casos pode diminuir o risco de um acidente.

Outro aspecto importante refere-se aos eventuais erros existentes num determinado sistema provocado por erros de seus projetistas. Embora os projetistas trabalhem sob condições de menor pressão que os operadores, eles também cometem erros. Os erros relacionados com os projetistas estão focados em aspectos como, dificuldade em alocar probabilidades em eventos raros, não consideração de efeitos colaterais, não atenção para medidas contingênciais, controle da complexidade do sistema, concentrando-se apenas em alguns

aspectos do problema, capacidade limitada de compreender relações complexas, além de dificuldades em se ter uma visão ampla do sistema, especialmente no que diz respeito aos sistemas críticos computacionais.

Desta forma, é importante a participação de operadores nestes sistemas. A questão que se deve discutir é qual deve ser o seu papel nestes sistemas

b) O papel do ser humano nos Sistemas Críticos quanto à Segurança

A automação de sistemas provoca o reposicionamento do ser humano em novos níveis de complexidade, com um maior nível de controle e supervisão, refletindo, desta forma, maiores níveis de tomada de decisões. Neste sentido, aumenta-se o nível de centralização, tornando a base de decisão extremamente mais complexa. Desta forma, o estudo da confiabilidade humana e da ergonomia passa a ter um papel fundamental dentro do estudo da segurança dos sistemas computacionais. Nessa relação do operador com os sistemas computacionais o ser-humano pode assumir basicamente três papéis: Monitor, Back-Up ou Parceiro.

A experiência demonstra que o ser humano apresenta um desempenho fraco como monitor em sistemas automatizados. Nesta situação o operador está extremamente dependente de informações fornecidas pelo sistema, que podem ser disponibilizadas em grande quantidade e de forma não adequada. Vale ressaltar, nessa situação, que as tarefas, que requerem baixa atividade do operador, podem implicar numa baixa vigilância de sua parte, podendo conduzi-lo a atitudes de super confiança ou complacência em relação ao sistema automatizado.

Quando o operador desempenha o papel de Back-up, o projeto do sistema de automação pode tornar-se de difícil gerenciamento durante uma situação emergência ou em sistemas complexos. Nestas situações o operador pode ter muito pouco tempo para decidir, além de o sistema oferecer diversas opções de escolha. Adiciona-se a isto, o fato do operador estar inativo por longos períodos, dificultando ainda mais sua tomada de decisão.

Já quando o operador tem o papel de Parceiro dentro do sistema crítico, ele irá realizar tarefas que não puderam ser automatizadas por algum motivo ou intencionalmente alocadas a ele. Neste modo de trabalho, podem surgir

algumas vantagens e desvantagens que devem ser avaliadas em cada caso. Podem surgir tarefas extremamente complexas e com muita arbitrariedade na tomada de decisão. Por outro lado, o operador pode agir com maior criatividade e utilizando seus conhecimentos a respeito do sistema, se sentindo, desta forma, realmente integrado no processo de automação.

Nesse sentido, o foco, com relação ao Operador, dentro um projeto de um sistema crítico quanto à segurança, deve ser sua Operação e não sua Função. O Operador deve ser envolvido num processo de tomada de decisões do projeto e da sua análise de risco. Quando o operador atua como parceiro, ele se sente realmente integrado e motivado para atuar no âmbito da responsabilidade deste tipo de sistema.

8. Metodologia de Análise de Risco Proposta

Neste capítulo é apresentada uma Metodologia de Análise de Risco para sistemas críticos quanto à segurança. Esta metodologia foi desenvolvida através da sinergia de dois aspectos fundamentais: a pesquisa acadêmica e a identificação de necessidades através de experiências práticas em análise de risco de sistemas críticos na área de transporte metroviário e aeroviário. No entanto, esta metodologia pode ser aplicada a outros sistemas críticos quanto à segurança através da avaliação e a adequação às características das outras áreas de aplicação. Trata-se de uma via de duas mãos. Tanto a pesquisa acadêmica abre novas possibilidades para a resolução de diversos problemas práticos, como a experiência auxilia em prover um maior direcionamento nas pesquisas acadêmicas sendo realizadas.

A metodologia, aqui apresentada, é parte integrante de um processo mais amplo denominado Análise de Segurança, constituído pelo Gerenciamento da Segurança e pela Análise de Risco, propriamente dita. O aspecto de gerenciamento da segurança não é o foco principal desta pesquisa.

Dentro da metodologia de Análise de Risco são destacadas as seguintes etapas:

- Definição e descrição do sistema, suas interfaces e demais informações necessárias para a Análise de Risco;
- Realização do processo de Análise de Perigo;
- Qualificação do Risco Residual;
- Avaliação da severidade dos acidentes relacionados com o estado perigoso; e
- Realimentação e Avaliação da experiência operacional.

No que se refere ao processo de Análise de Perigo são apresentadas as seguintes etapas:

- Análise Preliminar de Perigo;
- Análise de Perigo do Sistema;
- Análise de Perigo do Subsistema;
- Análise de Perigo da Operação e Suporte; e
- Análise Final de Perigo

A Análise de Risco pode ser exigida através de um processo denominado de **Certificação**. [Sotrey 96]

8.1. Métodos para Realizar Análise de Perigo

É apresentado, neste item, alguns métodos que podem ser utilizados durante o processo de Análise de Perigo. É importante destacar que muitos métodos devem ser utilizados em conjunto e serem avaliados por especialistas. Vale dizer também que a melhor cobertura obtida na Análise de Perigo é conseqüência da combinação dos diversos métodos. Evidentemente existem diversas interseções lógicas entre os métodos. No entanto, a complementariedade entre eles é que garante uma maior cobertura no processo de Análise de Perigo. [Profit 95]

8.1.1. Lista de Verificação

Esta técnica é derivada, normalmente, de normas, práticas de boa engenharia e da experiência adquirida através de diversos projetos. Esta lista de verificação pode ser uma referência para reflexão sobre o sistema que estiver sendo desenvolvido ou avaliado, devendo ser utilizada ao longo de todo o ciclo de vida do sistema.

Dentro do escopo da Análise de Perigo esta técnica pode ser utilizada na obtenção de maiores detalhes sobre os possíveis perigos, como também, na orientação de decisões de projeto, procurando, desta forma, diminuir os riscos envolvidos.

8.1.2. Árvore de Falhas

A técnica de Árvore de Falhas tem como objetivo fundamental estudar, em detalhes, a forma com que os perigos podem ser alcançados, através da combinação lógica de eventos primários. Neste sentido, a Árvore de Falhas não avalia a possibilidade de um novo perigo, mas estuda em detalhes os perigos já identificados.

O topo de uma Árvore de Falha constitui-se num perigo, ou até mesmo, em um acidente. Desta forma, a partir de um determinado Requisito Geral ou Específico de Segurança, adota-se como *Topo* desta árvore o evento da

negação do respectivo requisito. Assim sendo, essa técnica utiliza-se de uma filosofia top-down de detalhamento.

Na construção de Árvores de Falhas são utilizados os conectivos lógicos, na maioria das vezes OR e AND para manter a simplicidade do seu entendimento. A Árvore de Falhas é detalhada até se atingir um evento primário, ou básico, ou se chegar a um determinado evento que deverá ser detalhado posteriormente. Após a construção de uma Árvore de Falhas, ela pode sofrer dois processos de avaliação: qualitativo e quantitativo. [Kececioglu 91]

A avaliação qualitativa tem como finalidade representar a ocorrência do perigo através de uma forma lógica equivalente, mostrando a combinação dos eventos básicos que podem causar o evento de topo. Pode-se, através desta avaliação, determinar a criticidade dos eventos envolvidos, podendo inclusive influenciar em decisões de projeto ao longo do desenvolvimento.

A avaliação quantitativa tem como meta avaliar a probabilidade de ocorrência do evento de topo em função das probabilidades de ocorrência dos eventos básicos. Neste sentido é de fundamental importância verificar a independência dos eventos básicos envolvidos.

A título de ilustração é apresentado na figura 8.6 um exemplo simples de Árvore de Falhas com suas respectivas avaliações qualitativa e quantitativa.

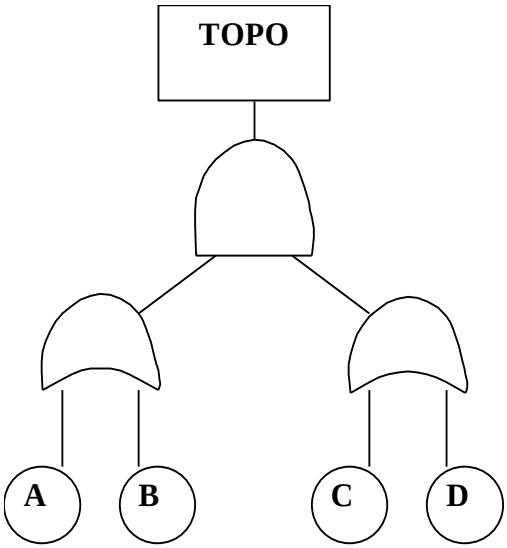


Figura 8.6 – Árvore de Falhas

Avaliação Qualitativa:

TOPO = (A + B) . (C + D)

A C + A D + B C + B D

Avaliação Quantitativa:

P (TOPO) = P (A) . P (C) + P (A) . P (D) + P (B) . P (C) + P (B) . P (D)

- P (A) . P (C) . P (D) - P (B) . P (C) . P (D) - P (A) . P (B) . P (C)

- P (A) . P (B) . P (D) + P (A) . P (B) . P (C) . P (D)

onde P(X) indica a probabilidade da ocorrência do evento X.

8.1.3. Árvore de Eventos

O método de Árvore de Eventos tem como principal meta avaliar quais eventos finais podem acontecer como conseqüência da combinação da ocorrência de eventos iniciais. Estes eventos iniciais podem se constituir em uma determinada falha de componente do sistema ou ocorrência de eventos externos. Para cada evento inicial devem existir, no mínimo, duas ramificações, uma com seu sucesso e outra com seu fracasso. Nesse sentido, este método tem como filosofia de detalhamento a técnica bottom-up, contrário ao método de Árvore de Falhas e, portanto, tem uma função complementar. Como para sua aplicação são necessárias maiores informações sobre eventos iniciais, este tipo de método é mais adequado de ser aplicado após o projeto detalhado. Na figura 8.7 é apresentada uma ilustração simplificada de aplicação deste método. Foram omitidas, nas probabilidades dos eventos, os termos (1 – Px), para efeito de simplificação.

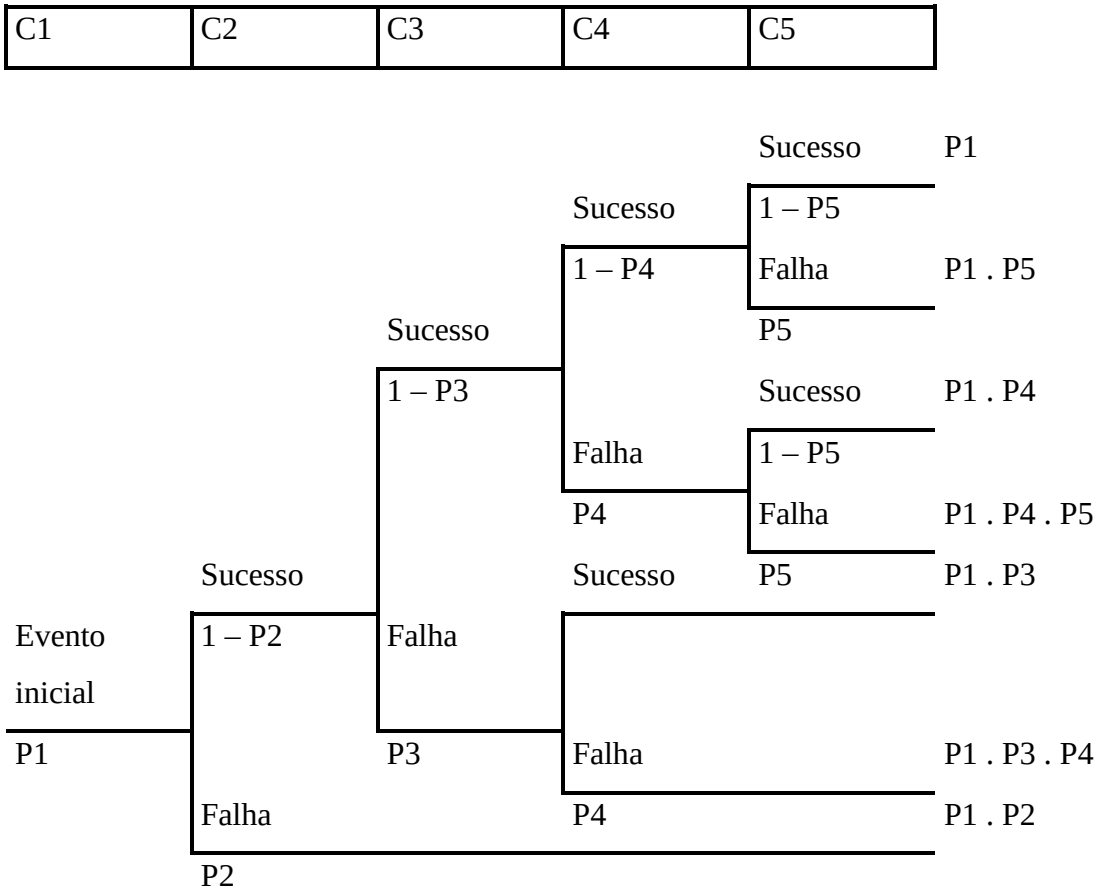


Figura 8.7 – Árvore de Eventos

8.1.4. Análise dos Efeitos dos Modos de Falhas - FMEA - Failure Modes and Effects Analysis

O método FMEA tem como objetivo identificar os efeitos dos modos de falha de um determinado componente do sistema. Este método pode ser aplicado sobre os níveis de sistema, subsistema ou módulos, como uma placa de hardware ou uma rotina de software. Quando é aplicado sobre o nível de sistema, os componentes a serem avaliados são os seus subsistemas constituintes. Quando o método é aplicado ao nível de subsistema os componentes são seus grandes módulos, sejam eles hardware ou software. Por outro lado, quando o método é aplicado sobre módulos específicos como, por exemplo, uma placa de hardware, os componentes referem-se aos seus elementos básicos, como um circuito integrado, um capacitor, um resistor, uma memória, entre outros. Esta última forma de aplicação tem sido a mais comumente utilizada. A aplicação do FMEA sobre módulos de software, apesar de possível, tem sido pouco utilizada, devido à grande complexidade de

sua aplicação em relação aos benefícios obtidos na qualidade do projeto. Na realidade, outros métodos são apresentados neste trabalho que melhor se aplicam à avaliação de módulos de software.

A tabela 8.5 apresenta um exemplo estrutural de uma tabela de FMEA que pode ser utilizada. Para cada componente considerado existe uma coluna que representa a sua probabilidade de falha. Em seguida existe outra coluna onde são detalhados todos os modos de falha possíveis de cada um dos componentes. Para cada um destes modos de falha são calculadas as probabilidades de sua ocorrência, seus efeitos locais e no sistema.

Componentes	Probabilidade de falha	Modos de falha	% por modo de falha	Efeitos **Locais**	Efeitos No Sistema	
					Critico	Não Crítico
A	1×10^{-3}	Aberto	90	Diminui Tensão		X
		Curto	5	Aumenta Tensão	5×10^{-5}	
		Outros	5	Aumenta Tensão	5×10^{-5}	

Tabela 8.5- Tabela de FMEA

8.1.5. Análise Crítica dos Efeitos dos Modos de Falhas - FMECA - Failure Modes, Effects, and Criticality Analysis

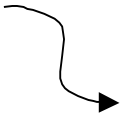
O método FMECA constitui-se na adição, ao método FMEA, de uma análise mais detalhada da criticidade da falha. Este estudo mais detalhado pode envolver a determinação dos meios de controle da falha ou até mesmo a necessidade de projeto de um controle adicional visando a diminuição do risco envolvido.

8.1.6. Redes de Petri

As Redes de Petri são especialmente apropriadas para representar sistemas com alto grau de concorrência. Os elementos de uma Rede de Petri são:

NÓ

TRANSIÇÃO



CAMINHOS.ARCOS

MARCAS - TOKEN

As Regras básicas de sua formação são:

- Uma transição é permitida quando todos os estados que possuem arcos orientados para a transição, contêm marcas;
- Quando uma transição é permitida, ela é disparada, retirando-se uma marca de cada um dos estados de entrada da transição e colocando-se uma marca em cada um dos estados para os quais existe um arco orientado que sai da transição;
- Um estado qualquer nunca pode estar ligado por outro arco orientado a outro estado; e
- Uma transição qualquer nunca pode estar ligada por um arco orientado a outra transição.

Na figura 8.9 é apresentado um modelo de um cruzamento de uma ferrovia com uma rodovia, através de Redes de Petri. Através de sua execução, pode ser gerado o Grafo de Alcançabilidade, conforme apresentado na figura 8.10. Através deste Grafo de Alcançabilidade podem ser identificados os estados perigosos (X X P3 X X P11 X X) e avaliados métodos de controle destes perigos.

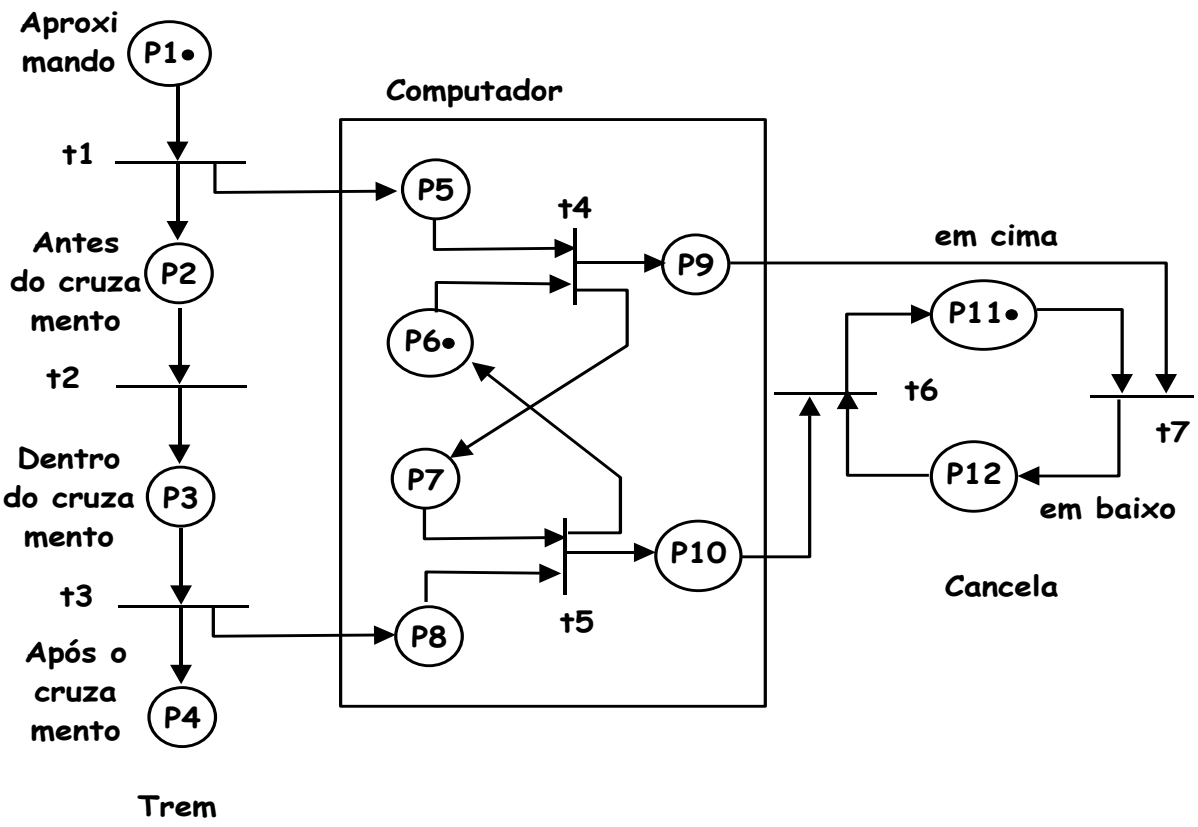


Figura 8.9 – Modelo de Cruzamento através de Rede de Petri

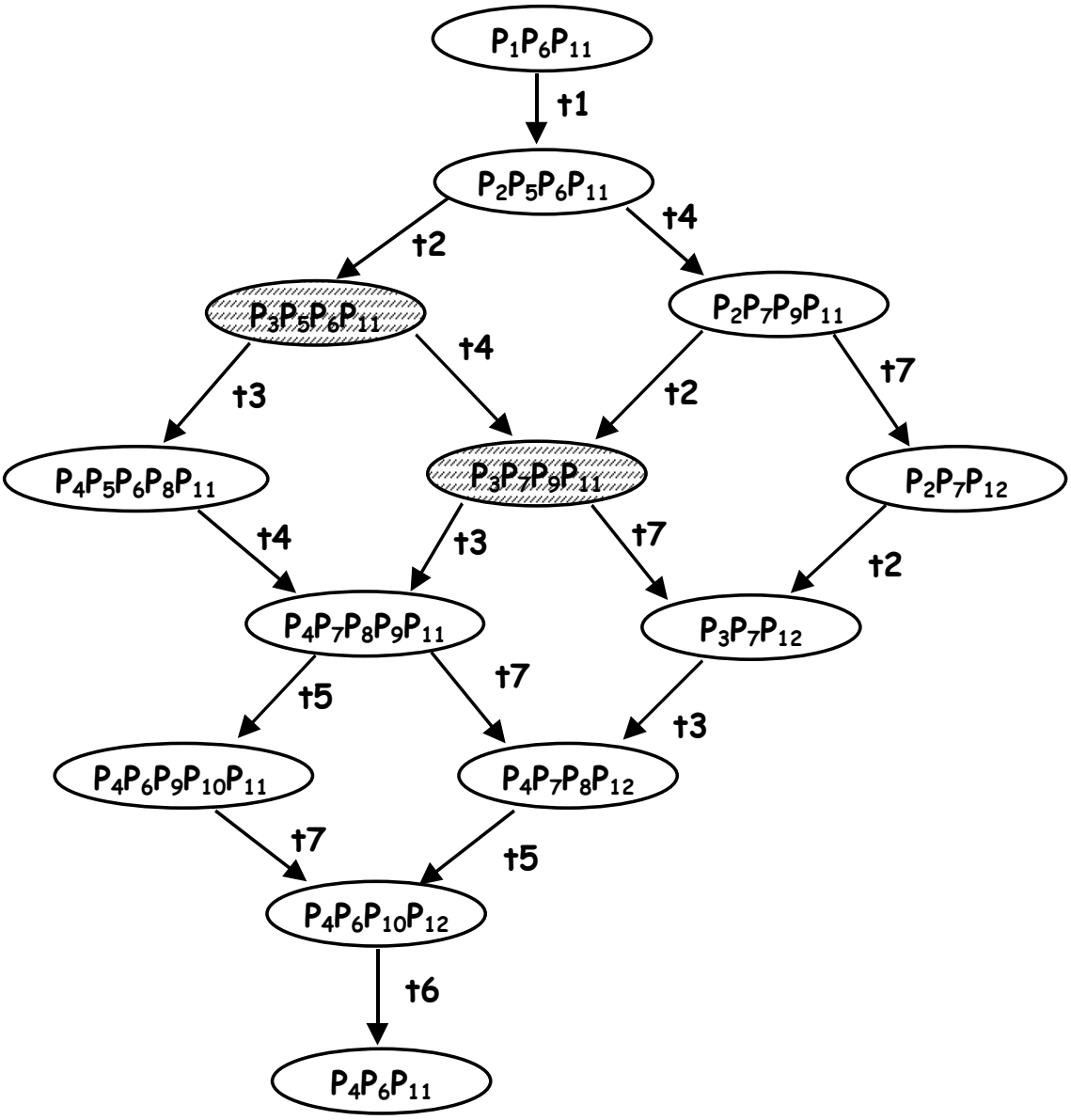


Figura 8.10 – Grafo de Alcançabilidade do Cruzamento através de Rede de Petri

9. Referências Bibliográficas

- Fault –Tolerant Systems. Israel Koren and C. Mani Krishna. Morgan Kaufmann Publishers, 2007.
- Reliability of Computer Systems and Networks. Fault Tolerance, Analysis and Design. Martin L. Shooman. John Wiley & Sons, 2002.
- Safety-Critical Computer Systems. Neil Storey. Addison-Wesley, 1996.
- Reliability Engineeing Handbook – Volume 2. Dimitri Kececioglu. Prentice Hall, 1991
- Design and Analysis of fault-Tolerant Digital Systems. Barry W. Johnson. Addison-Wesley, 1989.