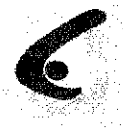

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE ENGENHARIA ELÉTRICA

ESTUDO DE CASO PARA A DESCRIÇÃO DE TÉCNICAS DE TOLERÂNCIA A FALHA EM VHDL

Aluno: Maí Correia Rabelo de Vasconcelos
Orientador(a): Maria de Fátima Queiroz Vieira Turnell
Disciplina: Projeto de Engenharia Elétrica - TCC
Período: 2006.2

Campina Grande, 10 Abril de 2007

Biblioteca Setorial do CDSA. Fevereiro de 2021.

Sumé - PB

Sumário

Capítulo 1

Introdução

1.1 Desafios da nanoeletrônica

Os avanços na eletrônica têm seguido a lei de Moore, através do escalonamento dos dispositivos, dobrando a capacidade de integração de transistores a cada dois anos, resultando nos complexos circuitos integrados da era atual.

Grande parte dos circuitos VLSI (*Very Large Scale of Integration*) de hoje são baseados na tecnologia CMOS (*Complementary Metal-Oxid-Semiconductor*) e o estado da arte no processo de fabricação já ultrapassa a escala de 90 nm. Porém, à medida que a tecnologia CMOS entra no domínio da nanoeletrônica, a produção de dispositivos passa a encontrar uma série de desafios como limitações impostas por flutuações térmicas, dissipação de potência e predominância de efeitos quânticos, ameaçando assim a taxa histórica de evolução que vem sendo presenciada [1, 2].

Um único CI (circuito integrado) moderno possui mais de um bilhão de transistores, milhares de interconexões estreitas e de contatos de metal. Os óxidos nos transistores têm diminuído e as dimensões são agora da ordem de 5-7 moléculas de SiO_2. Metais e óxidos possuem modos de falhas que sempre estiveram presentes, mas agora são muito mais significantes nas tecnologias atuais. Dentre os possíveis fenômenos que podem induzir falhas nos metais pode-se citar a *eletromigração*, que está ligada às densidades de corrente nos metais e aos níveis de temperatura de operação e o SIV (*stress-induced voiding*) descoberto na década de 80 onde algumas trilhas de metal podem se fragmentar formado circuitos abertos mesmo quando o CI não está alimentado. Quanto aos óxidos, a sua qualidade é fundamental para a operação correta do dispositivo, especialmente porque está ligado diretamente ao controle do canal nos transistores através do terminal porta (*gate*) e constitui um parâmetro importante na modelagem das capacitâncias parasitas implicando diretamente na máxima freqüência de operação dos dispositivos. Nesse sentido, uma série de *tradeoffs* envolvendo área, potência, custo, rendimento e confiabilidade são cada vez mais abordados na literatura e vêm exigindo conhecimentos mais aprofundados da ciência desses materiais.

As complexidades no processo de fabricação podem levar a defeitos físicos durante vários estágios de produção dos *chips*, incluindo a formação do cristal de silício, oxidação, difusão e litografia óptica. As tecnologias CMOS modernas, conhecidas como *deep-submicron technologies*, possuem um comportamento estatístico que levam à produção de dispositivos bons ou ruins em várias proporções, de acordo com o processo de ma-

turidade [3]. Dispositivos com defeitos em alguma etapa podem passar nos testes de qualidade e cair nas mãos do usuário final.

Quando um defeito ocorre em uma área ativa do circuito e produz dados incorretos, é dito que ocorre uma *falha*. É nesse contexto que destaca-se o projeto de *circuitos com tolerância a falha*.

1.2 A abordagem de tolerância a falha

Tolerância a falha é uma área de pesquisa tradicional que provém da necessidade de computação confiável e de operação sem interrupção em condições críticas e/ou hostis [2]. Um exemplo de extrema necessidade desse caráter tolerante são os circuitos de controle de satélites que ficam expostos à altos níveis de radiação e o seu custo de acesso à manutenção é bastante alto após ter sido posto em órbita. É também necessária a tolerância a falha em aplicações em que a probabilidade de erro deve ser reduzida ao máximo para evitar a tomada de decisões que levem à eventos catastróficos.

Diz-se que um circuito eletrônico é tolerante à falhas se ele é capaz de se recuperar e produzir uma saída correta mesmo quando parte desse circuito produziu um resultado incorreto.

Vários métodos diferentes têm sido desenvolvidos ao longo dos anos, mas todos eles possuem algo em comum - **redundância**. Estas podem ser redundâncias em recursos ou redundâncias temporais. Nas abordagens de hardware as técnicas de redundância em recursos são de maior interesse, e entre elas pode-se citar a NMR (*N-Modular Redundancy*), a multiplexação NAND de Von Neumann e a reconfiguração (estática ou dinâmica).

Neste trabalho será apresentado um estado da arte das técnicas de tolerância a falha, principalmente os trabalhos que tiveram como motivação à questão dos defeitos/falhas ligados ao processo de escalonamento dos dispositivos CMOS e será apresentado os resultados de um estudo de caso baseado na aplicação de arquiteturas tolerantes à falha de um sistema de aquisição de dados de um sensor de temperatura embarcado. Pode-se distinguir basicamente duas linhas de abordagem ao problema de falhas em circuitos: *mascaramento de erros* e *detecção e correção de erros*. Nos propomos aqui a implementar uma técnica de cada uma das linhas citadas, a TMR (*Triple Modular Redundancy*) dentro da proposta de mascaramento de erros e uso de códigos de Hamming na de detecção e correção de erros, para que sejam avaliadas as vantagens e desvantagens de cada método no contexto da aplicação escolhida.

A idéia do sistema escolhido foi concebida por J. M. M. Smith [4], porém, diversas alterações foram propostas na sua descrição original. A linguagem de descrição de hardware utilizada foi a VHDL e a ferramenta de síntese foi o QUARTUS II/ALTERA v4.0 para o FPGA EP1C20F400C7 da série *Cyclone*. O trabalho está organizado como segue:

- No Capítulo 2 são apresentados os conceitos de tolerância a falha, técnicas tradicionais e feito um estado da arte dos métodos abordados na literatura.
- O Capítulo 3 constitui-se das descrições do sistema adotado no estudo de caso e das técnicas implementadas.
- No Capítulo 4 são apresentados os resultados das sínteses dos circuitos, comparações feitas entre as técnicas e as conclusões do trabalho.

Capítulo 2

Tolerância a falha na eletrônica CMOS

2.1 Conceitos

As principais causas de falhas são problemas de especificação, componentes defeituosos, imperfeições de manufatura, fadiga dos componentes físicos além de distúrbios externos como radiação, interferência eletromagnética, variações ambientais (temperatura, pressão, umidade) e também problemas de operação [5].

Na definição de falhas considera-se ainda:

- Natureza: falha de hardware, falha de software, de projeto, de operação, ...
- Duração ou persistência: permanente ou temporária.
- Extensão: local a um módulo, global.
- Valor: determinado ou indeterminado no tempo.

O conceito de **dependabilidade** (*dependability*) é bastante importante e está relacionado à garantia de serviço proporcionado por um sistema. A dependabilidade tem como principais medidas:

1. Confiabilidade: Capacidade de atender à especificação dentro de certo período de tempo dentro de condições definidas, considerando à ocorrência de falhas ou interferências.
2. Disponibilidade: Probabilidade do sistema estar operando num instante de tempo determinado.
3. Segurança: Probabilidade do sistema não provocar dano mesmo na descontinuidade de suas funções (*safety*) ou proteção contra falhas maliciosas (*security*).

Dentre as medidas citadas acima, destaca-se a **confiabilidade** (*reliability*) que é a medida mais usada em sistemas que não toleram operação incorreta nem mesmo em pequenos instantes de tempo ou em sistemas em que a intervenção humana é inviável, como por exemplo na aviação ou na exploração espacial. Confiabilidade é uma medida probabilística, pois falhas são fenômenos aleatórios [5].

Existem ainda outros conceitos na área como performabilidade, mantenabilidade, testabilidade entre outros, que no presente trabalho não foram necessariamente explorados.

2.2 Defeitos, falhas e erros

Diversos autores têm se ocupado com conceitos básicos e nomenclaturas na área de tolerância a falha. A conceituação dos termos *defeito*, *falha* e *erro* ainda é um ponto de discordância na literatura.

De acordo com pesquisadores da área de informática e computação, um defeito (*failure*) é definido como um desvio da especificação. Defeitos não podem ser tolerados, mas deve ser evitado que o sistema apresente defeito. Diz-se que um sistema está em estado errôneo ou em erro se o processamento posterior a partir desse estado pode levar a defeito. Finalmente define-se falha (ou falta) como a causa física ou algorítmica do erro [5].

Deve-se notar a relação de tais conceitos com o processo de escalonamento dos dispositivos que presenciamos atualmente. À medida que os circuitos reduzem de tamanho, rendimento e confiabilidade estão se tornando problemas críticos. Normalmente tratados como independentes, esses dois parâmetros têm se tornado altamente correlacionados em estruturas nanométricas [2].

A confiabilidade (definida na seção 2.1) pode ser afetada por três tipos de falhas:

1. Falhas permanentes;
2. Falhas intermitentes;
3. Falhas transientes.

As falhas permanentes são devido à mudanças irreversíveis na estrutura do circuito. As intermitentes agem como curto-circuito ou circuito aberto em certas condições e as falhas transientes são de caráter aleatório, produzidas por *crosstalk*, ruído térmico, etc.

A maneira como as palavras tolerância a falhas, tolerância a defeitos, falhas permanentes, intermitentes, transientes, rendimento e confiabilidade se relacionam foi adotada para este trabalho como definida por Teixeira e Naviner [2], e é resumida como segue:

- A habilidade de realizar operações corretivas, apesar da presença de componentes defeituosos é chamada de tolerância a defeito.
- A habilidade do circuito se recuperar de uma falha intermitente e/ou transiente está ligada com a tolerância a falha.
- Tolerância a defeito é uma abordagem de melhora em rendimento enquanto tolerância a falha de melhora em confiabilidade.

2.3 Efeitos do escalonamento em dispositivos CMOS

O escalonamento de dispositivos é implacável na indústria microeletrônica desde que a miniaturização dos transistores permite operações mais rápidas e aumento da densidade de portas lógicas por unidade de *chip*. Existe uma importante barreira encontrada quando as dimensões mínimas vão abaixo de 0,5 μm pois novas propriedades físicas aparecem e as dificuldades no processo de manufatura tendem a aumentar a probabilidade de ocorrência de um defeito/falha, sendo essa uma grande motivação para o aprimoramento de técnicas tolerantes à falha à medida que a nanoeletrônica toma espaço.

Dispositivos com dimensões abaixo de 0,5 μm são chamados de dispositivos de canal curto[1] (*deep-submicron* ou *short-channel devices*). Essa redução têm sido em busca de criar dispositivos mais rápidos, *chips* mais densos e com menor potência. Em ambientes que contem energia ionizante, tais características dos transistores aumentam a probabilidade de SEUs (*Single Event Upsets*). Os efeitos de radiação são de interesse especial da comunidade aeroespacial.

Dentre os principais efeitos pode-se citar a modulação de comprimento do canal, injeção de portadores quentes (*hot carrier injection*) e corrente sub-limiar (*subthreshold current*).

2.3.1 Modulação de comprimento do canal

Modulação de comprimento do canal nos transistores aparece quando o dispositivo está em saturação e está relacionado com o efeito de estrangulamento do canal (*pinch-off*). Quando o dispositivo está em saturação o canal não mais "toca"o dreno e adquire uma forma assimétrica mais fina nas proximidade do dreno. A Figura 2.1 essa alteração na morfologia do canal.

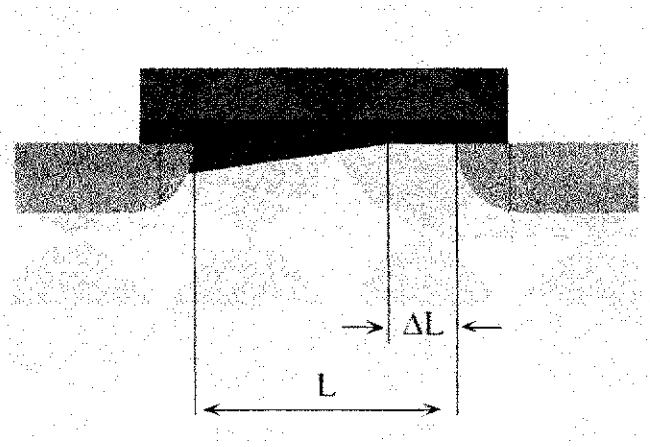

Figura 2.1: (Fonte: [6]) Efeito de modulação de comprimento do canal

A largura da camada de depleção que é responsável pela mobilidade dos portadores minoritários após o ponto de estrangulamento tende a aumentar com o aumento da tensão V_{DS} e o comprimento efetivo do canal diminui de ΔL (Fig. 1). Em dispositivos de canal longo, ΔL é desprezível quando comparado com L (comprimento total do canal), porém em dispositivos de canal curto essa largura se torna importante.

Dessa forma, mesmo após a saturação, a corrente I_D que atravessa o canal continua a aumentar (diferente do que acontece com os dispositivos de canal longo).

A Figura 2.2 apresenta uma comparação entre as curvas $I_D \times V_{DS}$ nas duas tecnologias.

Essa característica do crescimento da corrente após a saturação produz aumento significante na densidade de corrente presente no dispositivo, que podem levar a aumentar a *eletromigração* e vir a produzir **falhas** devido a vácuos ou extrusões na estrutura metálica. A eletromigração é um fenômeno bastante avaliado em diversas ferramentas de simulação

[1]Em contraste com os dispositivos de canal longo que possuem dimensões acima de 0,5 μm

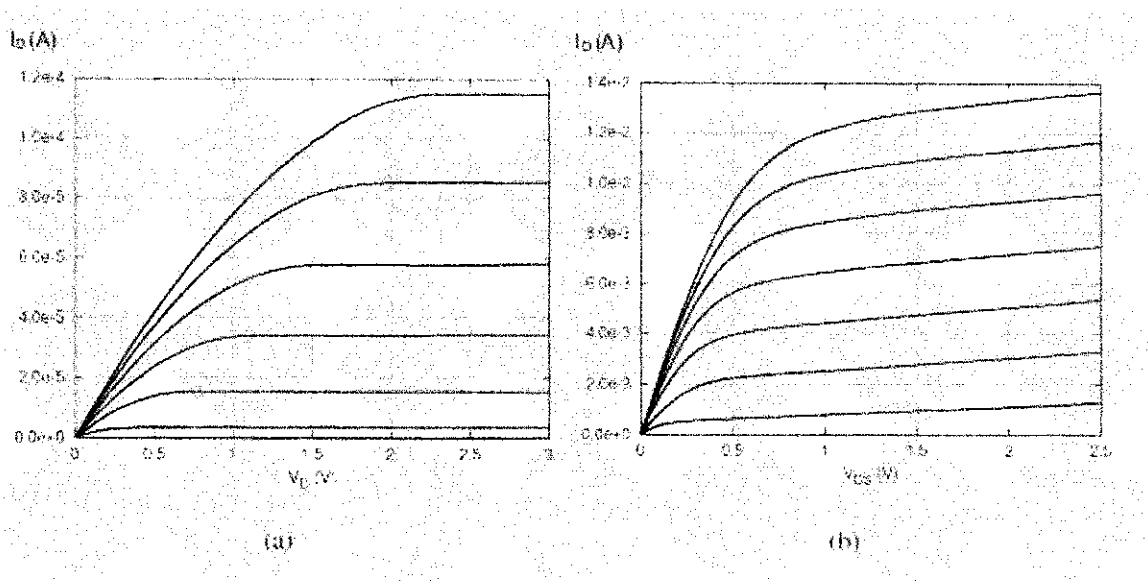

Figura 2.2: (Fonte: [6]) (a) Característica para canal longo, (b) Característica para canal curto.

de confiabilidade de circuitos [3] devido à sua alta propensão à gerar falhas com o processo de escalonamento dos transistores.

2.3.2 Corrente sub-limiar

A corrente sub-limiar (*sub-threshold*) é a corrente dreno-fonte que se estabelece mesmo quando a tensão na porta é menor que a tensão limiar (V_t) do transistor. A tensão limiar é o ponto de transição entre o estado de condução e de corte de um transistor. Observa-se que isso pode dar à tensão limiar uma definição vaga, já que no estado de corte espera-se que nenhuma corrente atravesse o dispositivo e estamos lidando com correntes sub-limiares. Porém isso pode ser explicado pelo fato da transição entre os estados de corte e condução se dar de forma contínua, ou seja, a variação dos portadores minoritários é exponencial e função da tensão de porta (V_{GS}).

Os transistores de canal curto das tecnologias *deep-submicron* emergentes possuem camada de óxido entre o substrato e o terminal porta extremamente fina (0,25 μm), operando com menores valores de V_t para garantir maior controle sobre a transição de estados e aumentar a freqüência de operação. Mas a redução desse valor aumenta a fuga de corrente devido à corrente sub-limiar durante o estado desligado do transistor.

Uma corrente sub-limiar em $V_{GS} = 0$ de 10 nA é insignificante em um único transistor, porém com os níveis de densidade de dispositivos por um unidade de área das novas tecnologias chegando à 100 milhões de transistores em um *chip* o impacto na potência total é significante [6].

As técnicas de tolerância a falha devem confrontar quantidade de redundância com o consumo de potência principalmente com esses níveis de corrente sub-limiar que são bastante apreciáveis com o escalonamento dos dispositivos.

2.3.3 Injeção de portadores quentes

Este fenômeno ocorre quando o campo elétrico do transistor na região de depleção dreno-canal é muito intenso. HCI (*hot carrier injection*) é uma falha sistemática que conduz à redução da freqüência máxima de operação.

HCI pode ser produzido pelo fato do canal ser muito curto ou por haver sobretensões de alimentação no dispositivo (observa-se que com os níveis baixos de tensão nas tecnologias *deep-submicron* quaisquer flutuações de tensão podem conduzir a HCI). Ou seja, dispositivos de escalas menores possuem características típicas para ocorrência do fenômeno.

HCI pode significativamente afetar a confiabilidade desde que alguns desses portadores com energia em excesso podem entram no óxido do terminal porta e causar danos na região de SiO_2, justificando mais uma vez a necessidade de aumento da confiabilidade através do projeto de circuitos com tolerância a falha.

2.4 Estado da arte das técnicas de tolerância a falha

Atualmente diversos esforços vêm sendo feitos no sentido de lidar da melhor forma possível com os *tradeoffs* envolvendo área, velocidade, potência e custo. As técnicas de tolerância a falha têm sido aperfeiçoadas na tentativa de reduzir ao máximo a redundância e ainda assim manter os níveis de confiabilidade exigidos nas especificações de projeto.

Um panorama geral mostrando algumas abordagens de tolerância a falha é apresentado na Figura 2.3.

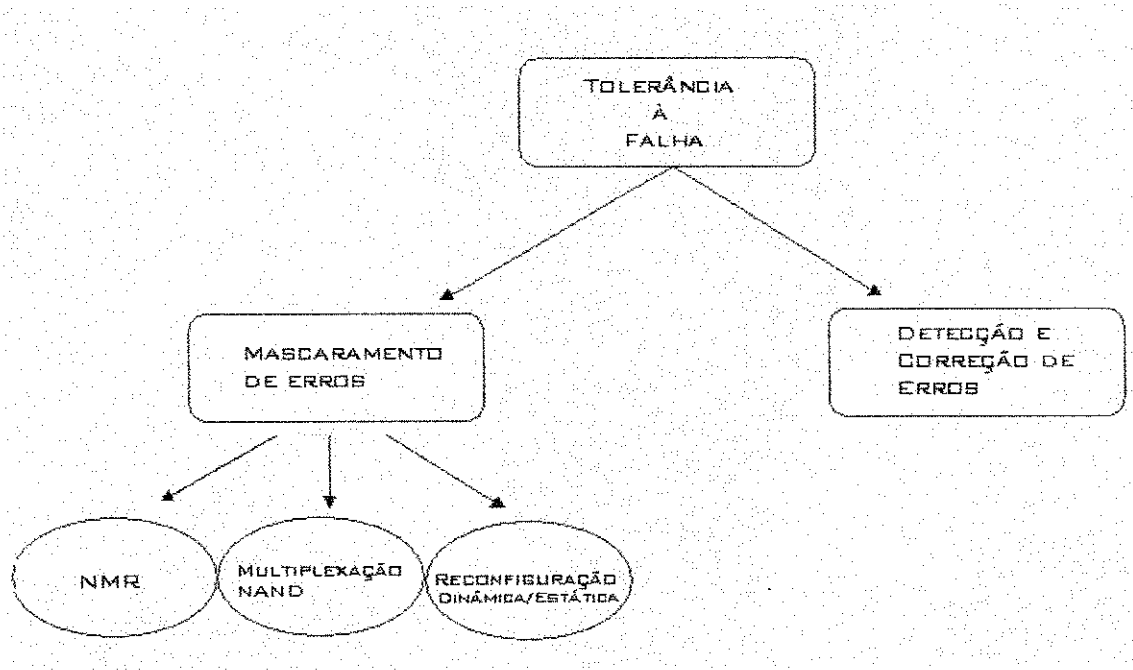

Figura 2.3: Linhas de pesquisa das abordagens de tolerância a falha

Observa-se que existem basicamente duas correntes. Uma delas abordando o problema através de técnicas de mascaramento de erros e outra fazendo a detecção e correção de

erros. A Figura 2.4 ilustra as configurações básicas das duas linhas.

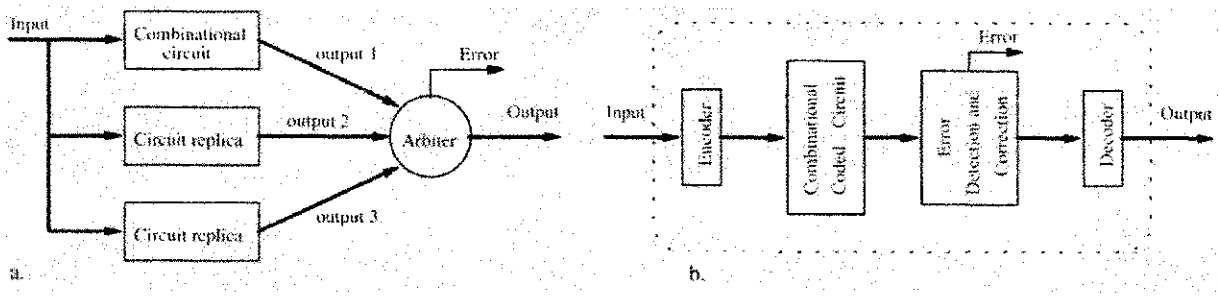

Figura 2.4: (Fonte: [2]) a. Mascaramento de erros b. Detecção e correção de erros

Dentre às técnicas de mascaramento de erro destacam-se à NMR (geralmente na forma de TMR), a técnica de multiplexação NAND de von Neumann e a reconfiguração, esta última pode ser estática ou dinâmica.

Han e Jonker [1] apresentam um esquema de tolerância a falha de grande potencial para a nanoeletrônica. A multiplexação NAND, iniciada por Von Neumann, é explorada e desenvolvida com níveis mais baixos de redundância. Foi identificado o comportamento estatístico das saídas de cada estágio de multiplexação NAND e sua evolução vista como uma cadeia de Markov. É mostrado que a técnica dificilmente funciona quando a taxa de erro do dispositivo lógico básico se aproxima de 0.1. Ficou claro nas conclusões do trabalho o *tradeoff* entre desempenho e redundância.

Em trabalho apresentado por Nikolic *et al* [7], é feita uma comparação de desempenho entre quatro esquemas: NMR, TMR em cascata, multiplexação NAND e a técnica de reconfiguração. Pôde ser visto no trabalho que a NMR e a multiplexação NAND são em geral piores que a reconfiguração, porém a reconfiguração se mostrou de pouca utilidade na proteção contra erros transientes, nesses casos a multiplexação NAND torna-se necessária.

Spica e Mak [8] levantam uma discussão interessante confrontando as abordagem de detecção e correção de erros (DCE) e a redundância, apostando que investir em melhorias nas técnicas de DCE seja mais adequado que explorar mais os métodos de redundância física.

Fox [9] desenvolve um método de correção de falhas considerando a propriedade dos circuitos *self-duals* e utilizando a técnica de inversão I/O (entrada/saída). Uma função é dita *self-dual* se a inversão de todas as suas entradas conduz ao complemento da saída original.

$$\overline{f(\overline{x})} = f(x) \tag{2.1}$$

Em seu trabalho são utilizados esquemas de somadores completos, multiplicadores e filtros elípticos na comparação entre o método proposto e a TMR. É mostrado que método proposto utiliza menor carga de lógica que a TMR e quantitativamente produz um uso de recursos de no máximo 64 % dos recursos exigidos na TMR nos três casos estudados.

Em grande maioria dos trabalhos o principal objetivo é mostrar que a técnica estudada é superior à alguma outra em termos de área, velocidade e potência. Esses são os

principais parâmetros de avaliação, e a relação que cada um tem com o outro leva sempre a *tradeoffs* no projeto de circuitos integrados.

De fato, dissipação térmica vem da energia liberada no chaveamento dos dispositivos e da energia necessária para conduzir os sinais elétricos nos circuitos. A energia mínima necessária para chavear um bit e a freqüência de chaveamento são limitadas pelo princípio da incerteza. Em outras palavras, o produto potência-atraso (*power-delay*) não pode ser menor que a constante de Planck, no limite quântico. Ou seja, a própria teoria quântica necessaria para a compreensão dos fenômenos em nanoescala indicam um *tradeoff* entre freqüência de clock e densidade, i.e., será necessário reduzir a freqüência de clock para altos níveis de densidade e níveis de densidade terão que ser reduzidos para altas freqüências de clock [10].

Espera-se que o sucesso dessas diversas técnicas sejam responsáveis pelo confiabilidade dos futuros computadores quânticos e possam trabalhar em conjunto com as promissoras tecnologias exóticas que vêm sendo descobertas como os nanofios, nanotubos, transistores moleculares, etc.

Capítulo 3

O sistema utilizado para aplicação das técnicas de tolerância a falha

3.1 Descrição do sistema

O sistema escolhido para a aplicação das técnicas de tolerância a falha foi obtido do livro de J.M.M. Smith - *Application-Specific Integrated Circuits* [4] e consiste em um esquema de aquisição de dados de um sensor digital de temperatura embarcado. A Figura 3.1 apresenta o esquema utilizado neste estudo.

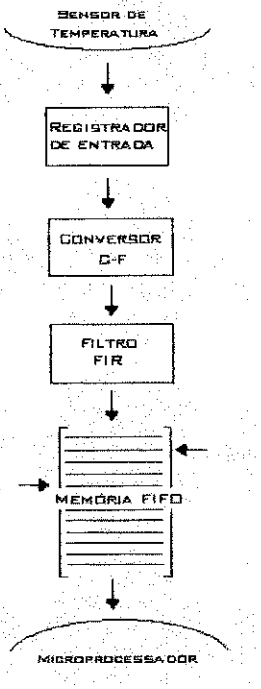

Figura 3.1: Esquema utilizado no estudo.

Os dados são inicialmente armazenados em um registrador de entrada, são então passados por um conversor que leva dados da escala Celsius para a escala Fahrenheit, em seguida por um filtro da média e são armazenados numa memória do tipo FIFO

assíncrona de forma que um microprocessador possa requisitar as informações segundo uma freqüência de clock independente da freqüência de clock da aquisição dos dados.

Para a descrição desse sistema foi usado a linguagem de descrição de hardware VHDL [11] e algumas alterações relativas às descrições originais foram propostas :

- Diferente da proposta original utilizou-se aritmética de ponto fixo durante toda a descrição.
- Todas as operações de divisão foram eliminadas para que as exigências das ferramentas de síntese fossem respeitadas e o desempenho fosse favorecido já que se trata de um sistema embarcado.
- A memória FIFO descrita opera de forma assíncrona enquanto a proposta original possui operação síncrona.
- Introduziu-se uma entidade denominada de `meta_filter_vector` que corresponde a um filtro de meta-estabilidade afim de sincronizar a comparação entre os contadores de escrita e de leitura na memória FIFO.

Nas subseções que seguem são descritos cada bloco do esquema escolhido.

3.1.1 Registrador de entrada

O projeto do registrador de entrada foi escolhido baseando-se nas especificações do sensor de temperatura digital LM73 da *National Semiconductor* [12]. Dessa forma, o registrador de entrada pode ter de 11 à 14 bits, sendo 9 bits para a parte inteira e uma precisão variando de 0.25 à 0,03125 para a parte fracionária.

Na descrição feita, as portas do registrador são: `Tr_in`,`Tr_out`, `clk` e `rst` correspondendo à temperatura de entrada, temperatura de saída, *clock* e *reset*, respectivamente.

Observe que a freqüência de *clock* define a taxa de aquisição de dados do sensor de temperatura.

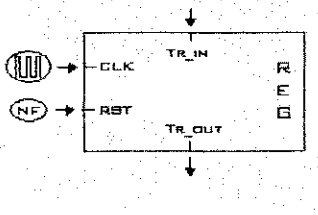

Figura 3.2: Registrador de entrada.

3.1.2 Conversor de temperatura

A função do conversor de temperatura neste esquema é de transformar os dados de temperatura da escala de graus Celsius (°C) para graus Fahrenheit (°F). A faixa de operação considerada é limitada pelo sensor, que no caso do LM73 é de no máximo -40 à 150 °C. Nessa faixa de operação utilizando a precisão máxima de 13 bits (0,0625 para parte fracionária), o erro máximo cometido na conversão é de 1,875 °F.

A conversão é dada pela expressão 3.1.

$$F = (\frac{9}{5})C + 32 \tag{3.1}$$

A aproximação proposta para o nível de precisão de 0.0625 permite que tal expressão seja escrita como

$$F = C + 0,5C + 0,25C + 0,0625C + 32 \tag{3.2}$$

Observa-se que os valores que multiplicam o dado de entrada na equação 3.2 são todos potências inversas de 2. Logo, para operar-se em aritmética de ponto fixo deve-se compatibilizar todos os termos da soma para a mesma base (Qj, em que j é o número de bits da parte fracionária), que no caso foi $Q4$.

Para isso, utilizou-se os operadores de deslocamento `srl` [13] quando o dado de entrada for positivo e `shift_right` [13] quando o dado for negativo representado em complemento de 2.

O Quadro 1 apresenta o a descrição em VHDL utilizada para o conversor de temperatura.

```
ENTITY conversor IS
      Port ( Temp_in : IN std_logic_vector (12 downto 0);
             Temp_out : OUT std_logic_vector (12 downto 0));
END conversor;

ARCHITECTURE arc_con OF conversor IS
k1: signed(12 downto 0) := "0001000000000";

BEGIN
process (Temp_in)
variable Tu_out: signed(12 downto 0);
variable A,B,C,D,E : signed(12 downto 0)

begin
A := signed(Temp_in);

if (Temp_in(12) ='1') then    -- Entrada negativa em complemento de 2
   E := A+1;
   B := shift_right(E,1); -- Operador de deslocamento aritimético
   C := shift_right(E,2);
   D := shift_right(E,4);
 else    -- Entrada positiva
    B := A srl 1;      -- Operador de deslocamento lógico
    C := A srl 2;
    D := A srl 4;
end if;

Tu_out := A + B + C + D + k1;    -- Conversão de temperatura
Temp_out <= std_logic_vector(Tu_out);
      end process;
end arc_con;
```

Quadro 1: Descrição do conversor de temperatura

3.1.3 Filtro FIR

O objetivo da etapa de filtragem é fazer uma compensação tanto dos erros de leitura do sensor como do erro cometido na conversão. O filtro utilizado foi um FIR de função de transferência

$$H(z) = \frac{1}{4}[1 + z^{-1} + z^{-2} + z^{-3}] \tag{3.3}$$

A equação de diferença correspondente é dada por

$$y[n] = \frac{1}{4}(x[n] + x[n-1] + x[n-2] + x[n-3]) \tag{3.4}$$

O filtro FIR também possui entradas associadas a *clock* e *reset* que são ligadas diretamente às entradas correspondentes no registrador de entrada.

Para a implementação do filtro em aritmética de ponto fixo utilizou-se simplesmente operadores de deslocamento para a compatibilização das bases nos termos de soma o que favoreceu bastante o desempenho da etapa de filtragem.

3.1.4 Memória FIFO

A implementação da memória FIFO foi feita de forma a permitir que dados saiam da fila com freqüência diferente da freqüência de entrada dos dados na mesma, constituindo assim uma FIFO assíncrona.

Observa-se então uma distinção entre dois domínios de *clock* (Figura 3.3) o que adiciona um nível extra de complexidade na otimização do algoritmo de síntese. Nota-se também que sistemas com múltiplos domínios de freqüência de operação exigem do projeto *overhead* de área e de velocidade na medida em que filtros de meta-estabilidade são necessários e é preciso tempo extra para cruzar os domínios distintos na comparação de variáveis.

A entidade que representa memória FIFO descrita em VHDL é apresentada no Quadro 2.

```
ENTITY fila IS
generic (width : integer := 5);

port ( rd_clk_reset    : in std_logic;
       wr_clk_reset    : in std_logic;
       rd_clk      : in std_logic;
       wr_clk      : in std_logic;
       rd_data     : out std_logic_vector (12 downto 0);
       wr_data     : in std_logic_vector (12 downto 0);
       full        : out std_logic;
       empty       : out std_logic);

END fila;
```

Quadro 2: Entidade `fila` descrita em VHDL.

Sendo as portas da entidade `fila` relativas a:

1. `rd_clk_reset`: controle de (*reset*) do domínio de *clock* de leitura.
2. `wr_clk_reset`: controle de (*reset*) do domínio de *clock* de escrita.

3. `rd_clock`: *clock* de leitura de dados da memória FIFO.
4. `wr_clock`: *clock* de escrita de dados na memória FIFO.
5. `rd_data`: dado de temperatura de leitura.
6. `wr_data`: dado de temperatura de escrita.
7. `full` e `empty`: *flags* de informação de memória cheia e vazia, respectivamente.

A Figura 3.3 apresenta a memória FIFO, suas entradas e saídas.

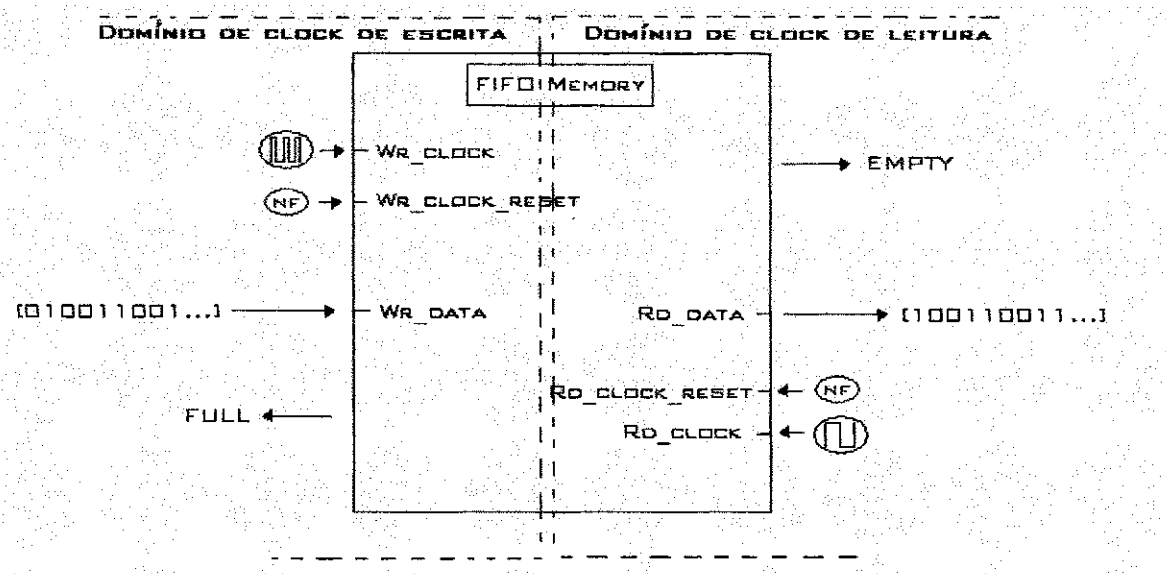

Figura 3.3: Entidade de descrição da memória FIFO.

Além das portas de entrada e saída da entidade, diversos sinais (*signals*) são utilizados na descrição em VHDL. Dentre esses sinais, destacam-se o `rd_counter` e o `wr_counter` que são os ponteiros de endereço de memória de leitura e escrita.

A estratégia utilizada para a identificação dos estados de fila cheia (*full*) ou vazia (*empty*) foi de adicionar um bit extra como um flag à representação binária dos endereços de memória já que quando os dois ponteiros são iguais a fila pode tanto estar cheia como estar vazia.

Por exemplo, se a FIFO contém 16 espaços para armazenamento de dados então seriam necessários ponteiros de 4 bits. Dessa forma, utiliza-se ponteiros de 5 bits (o mais significativo como um *flag*) seguindo os seguintes critérios:

- Se os bits mais significativos dos dois ponteiros são iguais, então a FIFO está vazia, ou seja, o ponteiro de escrita deslocou-se o mesmo número de vezes que o ponteiro de leitura.
- Se os bits mais significativos dos dois ponteiros são diferentes, então a FIFO está cheia, ou seja, o ponteiro de escrita deslocou-se um ciclo inteiro à mais que o ponteiro de leitura.

Observa-se então uma FIFO de comprimento variável, como era de se esperar para operações assíncronas.

Surge então o problema da comparação entre os ponteiros, ou seja, como deve-se comparar dois ponteiros que estão alocados em dois diferentes domínios de *clock*. A Figura 3.4 apresenta o problema da meta-estabilidade[1] nos flip-flops no momento da comparação entre os contadores.

Observa-se que a comparação nesse instante de tempo levaria a um erro já que o estado do sinal `wr_counter` (contador de escrita) estaria indefinido na transição.

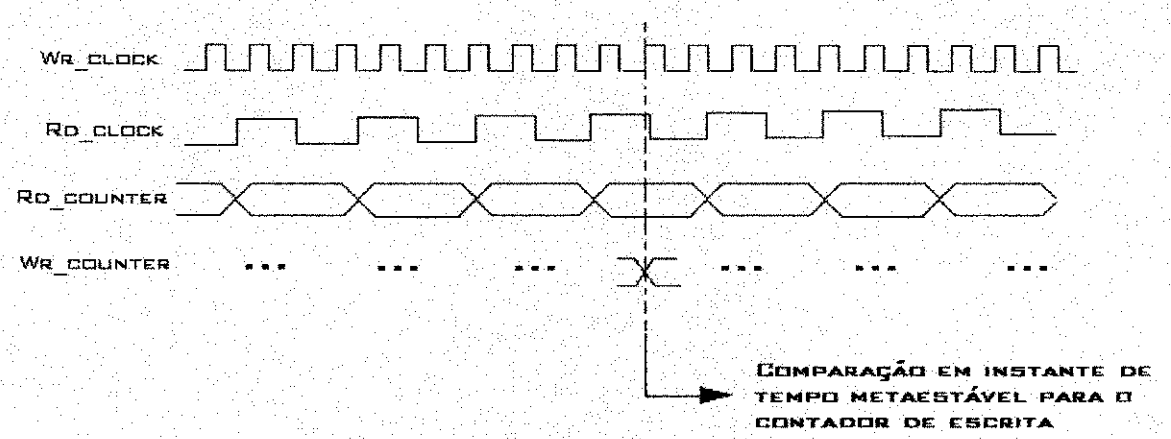

Figura 3.4: O problema da metaestabilidade na comparação dos ponteiros da FIFO.

Deve-se ter uma estratégia para resolver tal problema, visto que, é importante que a ferramenta de síntese a ser utilizada produza resultados a nível RTL e à nível de portas de forma que o circuito sintetizado em um FPGA não corra risco de perder o controle sobre os contadores da memória fila. Isso acarretaria um prejuízo grande para o funcionamento do sistema como um todo.

A solução adotada foi a utilização do contador em código Gray [14]. Com esse código é garantido que somente um bit da palavra do contador sofre alteração por período de clock, dessa forma valores não determinísticos no momento da comparação são evitados e mesmo que se perca o bit, no ciclo subseqüente ele será recuperado.

O algoritmo para implementação dessa solução utiliza um filtro de metaestabilidade [14], que nesse caso nada mais é que um registrador que pode operar sob dois domínios de clock distintos, recebe dados em código Gray e entrega na saída código binário sincronizando assim `rd_counter` com `wr_clock` e `wr_counter` com `rd_clock`. O algoritmo é descrito abaixo.

1. Converta os contadores de código binário para código Gray;
2. Envie o contador codificado com Gray através do filtro de metaestabilidade para o outro domínio de clock;
3. No outro domínio de clock, converta novamente para binário;
4. Faça a comparação entre os contadores.

[1] Os flip-flops estão predispostos a um problema chamado de metaestabilidade, que ocorre quando um dado ou uma entrada de controle está mudando no momento do pulso de clock.

O Quadro 3 exemplifica o processo dentro da descrição da arquitetura da memória FIFO, sensível à clock de escrita e reset de escrita (`wr_clock` e `wr_clock_reset`).

```
process(wr_clk,wr_clk_reset) begin
      if wr_clk_reset = '0' then
          wr_counter <= (others => '0');
      elsif rising_edge(wr_clk) then
          if full_in = '0' then
              wr_counter <= wr_counter_plus_1;
      end if;
if ((rd_cross_counter(3 downto 0) = wr_counter(3 downto 0)) and
(rd_cross_counter(4) /= wr_counter(4))) or
((wr_counter_plus_1(3 downto 0) = rd_cross_counter(3 downto 0))and
(rd_cross_counter(4) /=wr_counter_plus_1(4))) then
      full_in <= '1';
else
       full_in <= '0';
      end if;
  end if;
end process;
```

Quadro 3: Comparação entre contadores de leitura e escrita.

O sinal `rd_cross_counter` é o contador de leitura após passar pelo filtro de metaestabilidade em código gray e ser reconvertido para código binário. Observe que no exemplo é utilizado um contador de 4 bits + 1 bit de *flag*. Veja que basta que o próximo valor do contador de escrita (`wr_counter_plus_1`) seja igual ao de leitura (`rd_cross_counter`) que o estado de `full` é lançado.

3.2 A abordagem de mascaramento de erros: TMR

A técnica básica de redundância utilizada como estratégia de tolerância a falha foi a redundância modular tripla (TMR). O sistema mostrado na Figura 2.4 (seção 2.4) consiste em três circuitos digitais paralelos, todos com a mesma entrada. O elemento denominado *árbitro* (*arbiter* ou *voter*) é responsável por verificar a opinião majoritária e produzir a saída final do sistema. Observe que se um dos módulos produz um resultado errôneo e os outros dois o produzem corretamente, o árbitro é capaz de fornecer uma saída correta. Ou seja, considerando que o árbitro está livre de falhas e os módulos sendo identificados por A, B e C, a confiabilidade R do sistema é dada por

$$R = P(A \cdot B + A \cdot C + B \cdot C) \tag{3.5}$$

Seja p a probabilidade de sucesso de um módulo sendo essa idêntica para os três, em termos de teorema binomial tem-se

$$R = \binom{3}{3} p^3(1-p)^0 + \binom{3}{2} p^2(1-p)^1 = 3p^2 - 2p^3 \tag{3.6}$$

Admitindo uma distribuição temporal para p do tipo $p = e^{-\lambda t}$, então

$$R = 3e^{-2\lambda t} - 2e^{-3\lambda t} \tag{3.7}$$

Uma comparação pode ser feita entre as funções de confiabilidade do sistema sem aplicação da TMR e com a aplicação da mesma. A Figura 3.5 apresenta as curvas.

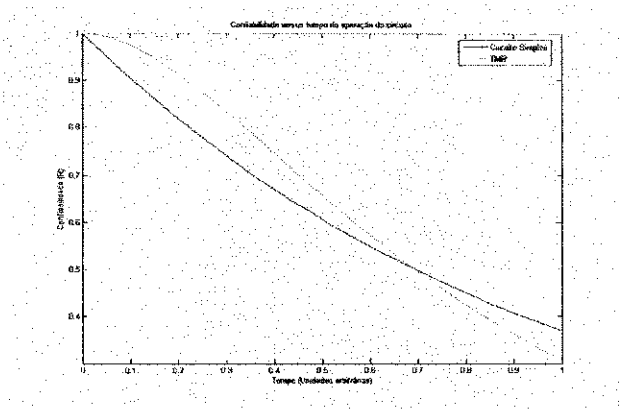

Figura 3.5: Curvas de confiabilidade versus tempo.

Observa-se que a TMR garante maior confiabilidade ao sistema para uma larga faixa de vida útil dos dispositivos e que a técnica gera bons resultados para níveis relativamente altos de confiabilidade de cada módulo individual.

Caso sejam utilizados mais módulos (4,5,6 ...) as curvas tendem a melhorar ainda mais a confiabilidade do sistema, porém às custas de *overhead* de área. A TMR além de ser um *benchmark* dentre os métodos de tolerância a falha é bastante justificável em aplicações em que confiabilidade é um parâmetro prioritário.

3.2.1 Descrição da TMR implementada no sistema

A técnica da TMR foi aplicada ao sistema descrito na seção 3.1. Os módulos foram triplicados e um diagrama de blocos da nova arquitetura é apresentado na Figura 3.7.

Neste estudo, os módulos árbitros são considerados livres de falhas e a inserção das falhas foi feita à nível modular, ou seja, ao invés das três entradas serem idênticas em cada um dos módulos, uma delas é forçada a gerar um resultado errado externamente para que seja possível verificar a funcionalidade da técnica. A Figura 3.6 apresenta o modelo de injeção de erros utilizado.

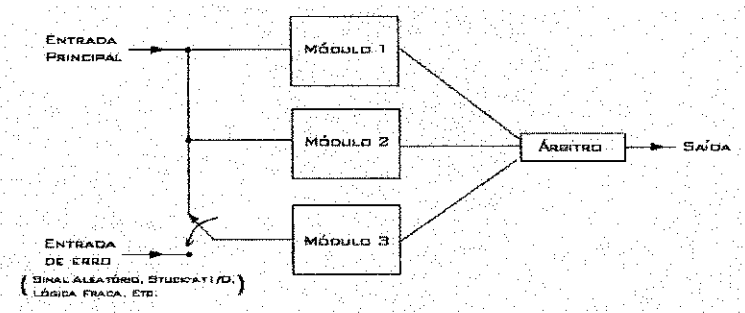

Figura 3.6: Modelo de inserção de falha. (Exemplo para o módulo 3)

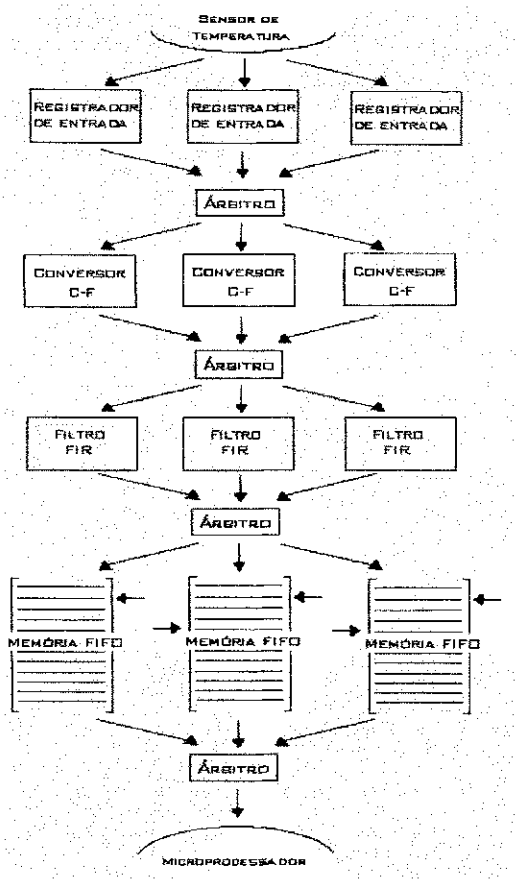

Figura 3.7: Sistema com arquitetura TMR tolerante à falha.

Uma característica desse modelo é a facilidade de implementação do árbitro, dado que basta-se criar um circuito comparador de três entradas e verificar qual resposta é majoritária. O Quadro 4 é a descrição feita em VHDL do árbitro.

```
ENTITY arbitro IS
      Port ( A : IN std_logic_vector (12 downto 0);
       B : IN std_logic_vector (12 downto 0);
       C : IN std_logic_vector (12 downto 0);
        result : OUT std_logic_vector (12 downto 0));
END arbitro;

ARCHITECTURE arc_arb OF arbitro IS
BEGIN
      process (A,B,C)
       begin
          for i in 0 to 12 loop
         result(i) <= (A(i) and B(i)) or (A(i) and C(i)) or (B(i) and C(i));
         end loop;
       end process;

END arc_arb;
```

Quadro 4: Implementação em VHDL do módulo árbitro.

Um método simples porém bastante eficiente de diagnóstico de erro foi também implementado através da comparação da saída produzida pelo árbitro e as suas entradas, sendo então possível identificar em que módulo ocorreu o erro. Tal diagnóstico é particularmente interessante em abordagens que utilize reconfiguração já que conhecendo-se o módulo com problema, a área do circuito em que o mesmo foi sintetizado pode ser isolada. O erro é apresentado na forma de um vetor de bits – `e(i)` – de tamanho 18 sendo os índices relativos a:

- `e(1) e(2) e(3)` - Registradores de entrada 1, 2 e 3.
- `e(4) e(5) e(6)` - Conversores 1, 2 e 3.
- `e(7) e(8) e(9)` - Filtros 1, 2 e 3.
- `e(10) e(11) e(12)` - Memoria FIFO1, Memoria FIFO2 e Memoria FIFO3.
- `e(13) e(14) e(15)` - *flag* `full` (FIFO1), *flag* `full` (FIFO2) e *flag* `full` (FIFO3)
- `e(16) e(17) e(18)` - *flag* `empty`(FIFO1) da FIFO3, *flag* `empty`(FIFO1) e *flag* `empty`(FIFO3).

Por exemplo,o valor de `e` $= (3FFFD)_{16}$ indica um erro no módulo 2 de registradores de entrada.

3.3 A abordagem de detecção e correção de erros: Códigos de Hamming

O **código de hamming**[2] é classificado como um código corretor linear binário que pode detectar e corrigir erros em um único bit, podendo detectar (mas não corrigir) até dois bits com erro.

O algoritmo geral para codificação de uma seqüencia de bits $d_1d_2d_3d_4...d_d$ em uma palavra código $f_1f_2f_3f_4...f_f$, com $(f > d)$ pode-ser resumido da seguinte forma[15]:

1. Todas as posições que são potências de 2 são reservadas à bits de paridade, $f_1, f_2, f_4, f_8, ...$
2. Todas as outras posições são para os dados a serem codificados, $f_3, f_5, f_6, f_7, f_9, ...$
3. Cada bit de paridade assume o valor do cálculo da paridade resultante dos bits que na representação binária da sua posição na palavra código possuem 1 na posição relativa ao bit de paridade. Em outras palavras, o bit de paridade na posição 2^k "checa"bits em posições que possuem em sua representação binária de posição, $k = 1$. Por exemplo, f_1 assume o valor d cálculo da paridade dos bits $f_3(11)_2, f_5(101)_2, f_7(111)_2, f_9(1001)_2...$).

No descodificador é feito o cálculo da paridade para cada bit de redundância. Caso todas as verificações resultem corretamente, significa que não houve erro na transmissão. Havendo um único erro, o cálculo da paridade indicará falhas no cálculo relativo a um ou mais bits de paridade, podendo-se assim identificar a posição do bit que ocorreu falha e através do complemento do mesmo fazer a correção da palavra.

A Tabela 1 apresenta as posições checadas pelos bits de paridade nas posições 1,2,4 e 8.

[2]Introduzido por Richard Hamming em 1950 quando trabalhava no *Bell's Laboratory*

Posição	1	2	3	4	5	6	7	8	9	10	11	12
Rep.binária	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100
Check1	X		X		X		X		X		X	
Check2		X	X			X	X			X	X	
Check4				X	X	X	X					X
Check8								X	X	X	X	X

Tabela 3.1: Posições de checagem dos bits de paridade

Sob o ponto de vista de teoria da informação o problema foi formulado por Hamming da seguinte forma:

Dado uma seqüencia binária de $\boldsymbol{d}$ bits representando a informação a ser codificada, o conjunto $\boldsymbol{D}$ é definido como o espaço de todas as possíveis mensagens a serem enviadas, possuindo assim 2^d elementos. A mensagem é codificada adicionando-se bits de paridade em palavras de $\boldsymbol{f}$ bits ($\boldsymbol{f} > \boldsymbol{d}$) dentro do conjunto $\boldsymbol{F}$. Os conjuntos $\boldsymbol{D}$ e $\boldsymbol{F}$ são então espaços vetoriais sob os corpos binários {1,0}.

A codificação, isto é a operação que consiste em transformar a mensagem de $\boldsymbol{D}$ para $\boldsymbol{F}$ pode ser vista como uma aplicação linear de $\boldsymbol{D}$ em $\boldsymbol{F}$ da forma $\varphi\{\cdot\}$, em que todos os resultados da álgebra linear se aplicam aqui.

Observa-se que o conhecimento da imagem da **base canônica** determina inteiramente a aplicação de codificação φ. Por exemplo, a base canônica para $\boldsymbol{d} = 4$, é o conjunto $d_1 = 0001$, $d_2 = 0010$, $d_3 = 0100$, $d_4 = 1000$.

Define-se a matriz de φ, **G**, chamada de matriz geradora, que é formada por $\boldsymbol{g}$ colunas, em que $\boldsymbol{g}$ é o número de bits da palavra de dados, sendo cada coluna correspondente à imagem de cada elemento da base canônica aplicada em $\varphi\{\cdot\}$. Por exemplo, num código de Hamming(7,4) a palavra código é da forma: **CCDCDDD** sendo C bits de redundância e D bits de dados. A aplicação linear leva a $\varphi\{0001\} = 1101001$, $\varphi\{0010\} = 0101010$, $\varphi\{0100\} = 1001100$, $\varphi\{1000\} = 1110000$. Observe que o valor dos bits de paridade é calculado através do cálculo da paridade dos bits que são checados pelo mesmo. A matriz **G** toma a forma:

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Dispondo-se de tal matriz, qualquer palavra código pode ser gerada pela multiplicação:

$$\mathbf{F'} = \mathbf{G} \cdot \mathbf{D'} \tag{3.8}$$

em que **F'** é a palavra código na forma de matriz coluna e **D'** é a palavra de dados na forma de matriz coluna sendo que as somas envolvidas na multiplicação matricial são feitas em módulo 2.

Ou seja, tendo-se a matriz geradora é possível gerar palavra código para qualquer seqüencia de dados do conjunto **D**.

Para a detecção e correção do erro, utiliza-se a chamada **matriz de contrôle (H)** que possui f linhas, sendo f o número de bits de paridade da palavra código e d colunas, em que d

é o número de bits da palavra código. As linhas são preenchidas "setando"como 1 as posições checadas por cada bit de paridade, inclusive, e como "0"os outros elementos. No exemplo anterior, a matriz **H** toma a forma:

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Sendo a primeira linha relativa ao bit de paridade da posição 4, a segunda o da posição 2 e a terceira o da posição 1, respectivamente na palavra código.

Ocorrendo apenas um erro, o produto das matrizes $\mathbf{H} \cdot \mathbf{C}$ é igual à um vetor coluna que indica a posição do erro em representação binária. Ou seja, não havendo erro $\mathbf{H} \cdot \mathbf{C} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$.

3.3.1 Descrição do código de hamming implementado no sistema

A implementação da técnica de detecção e correção de erros através do código de hamming consistiu na introdução de módulos de codificação e descodificação, havendo também a presença de módulos de inserção de erro. Para cada etapa do sistema de aquisição de dados, i.g. registrador de entrada, conversor, filtro FIR e memória fila, tais módulos foram acrescentados obtendo-se um esquema típico de transmissão de dados através de um canal. Como apresentado na Figura 3.8, no contexto do problema os canais são os módulos da descrição original.

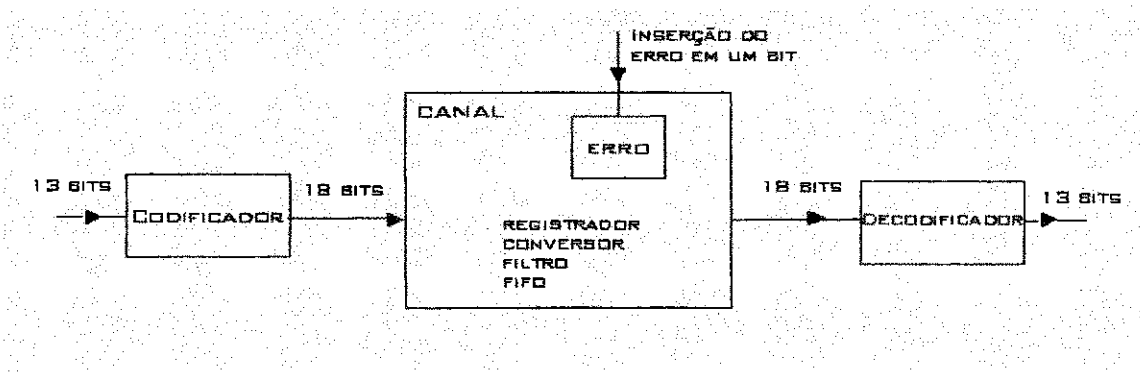

Figura 3.8: Esquema de codificação e descodificação utilizado.

Considere aqui o caso da palavra de dados possuir 13 bits. Sendo assim, faz-se necessário a introdução de mais 5 bits de paridade para a codificação de hamming, ou seja, a palavra código é da forma:

$$\mathbf{B}\mathbf{B}C_{16}\mathbf{B}\mathbf{B}\mathbf{B}\mathbf{B}\mathbf{B}\mathbf{B}C_8\mathbf{B}\mathbf{B}\mathbf{B}C_4\mathbf{B}C_2C_1$$

A matriz geradora **G** para uma paridade par é obtida através da aplicação linear sobre os elementos da base canônica, e tem a forma apresentada abaixo.

$$\mathbf{G} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

E a matriz de controle **H** possui a forma:

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Tendo essas duas matrizes é possível projetar os codificadores e descodificadores utilizando operações de ou exclusivo (XOR) em VHDL, produzindo o efeito da multiplicação das matrizes. O Quadro 5 apresenta trecho do código do módulo "codificador". É mostrado somente as operações que levam à obtenção dos bits de paridade na palavra código.

```
process(input,paridade)
BEGIN
        codeword(1) <= input(0) xor input(1) xor input(3) xor
            input(4) xor input(6)
            xor input(8) xor input(10) xor input(11)
            xor  paridade;
        codeword(2) <= input(0) xor input(2) xor input(3) xor
            input(5) xor input(6) xor input(9) xor
            input(10) xor input(12) xor paridade;
        codeword(4) <= input(1) xor input(2) xor input(3) xor
            input(7) xor input(8) xor input(9)
            xor input(10) xor paridade;
        codeword(8) <= input(4) xor input(5) xor input(6)
            xor input(7) xor input(8) xor
            input(9) xor input(10) xor paridade;
        codeword(16) <= input(11) xor input(12) xor paridade;
end process;
```

Quadro 5: Codificador hamming descrito em VHDL

3.4 Rotinas de teste

As ferramentas utilizados neste trabalho para os testes das descrições dos códigos em VHDL foram o simulador ALTERA/MODELSIM SE 6.1B [16] e a ferramenta de síntese da QUARTUS II V4.0 também da Altera. A seguir são apresentadas simulações considerando a inserção de erros no sistema para que as técnicas de tolerância a falha fossem validadas.

3.4.1 Testes - TMR

Nesta seção serão apresentadas simulações de inserção de falhas em módulos da TMR para cada estágio do sistema através de diagramas de tempo.

Caso 1: Falha tipo *stuck-at1* e presença de lógica fraca

(Etapa: `registrador_de_entrada`)

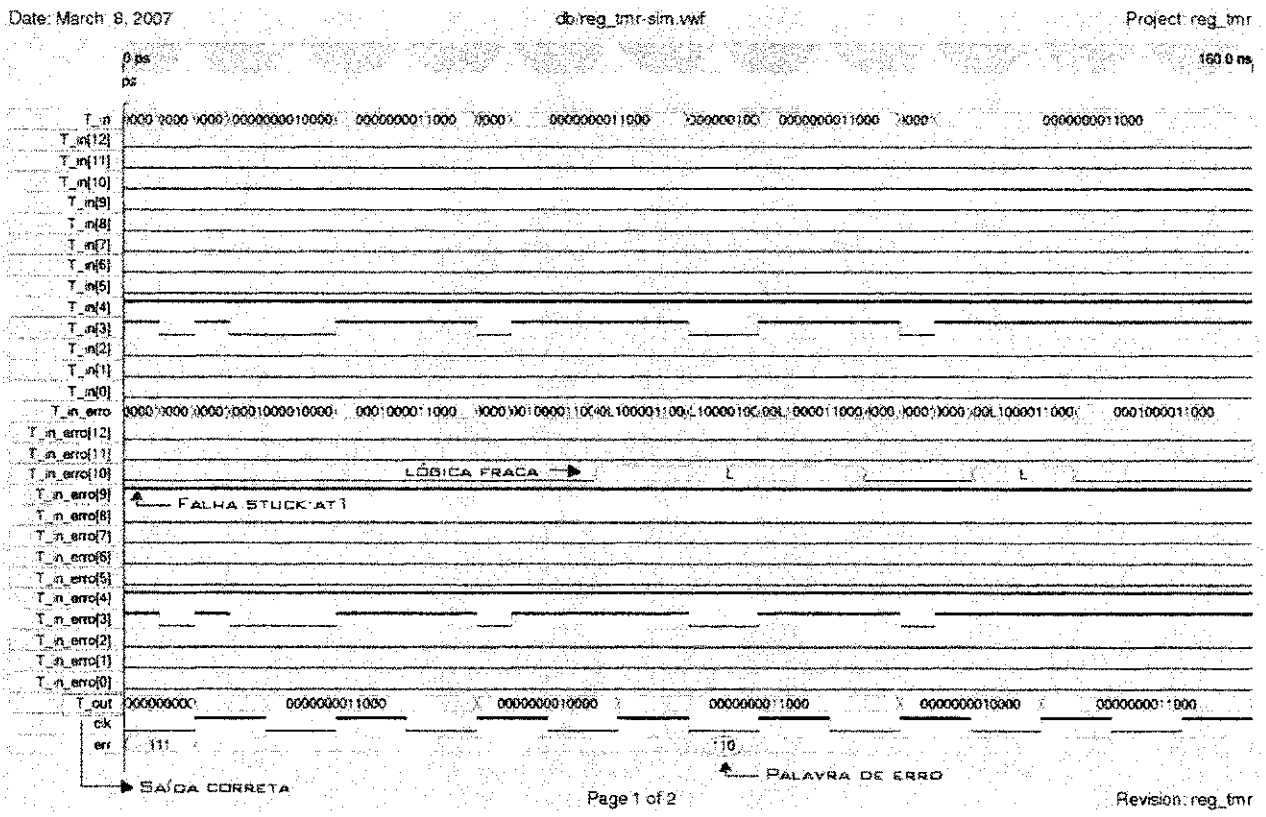

Figura 3.9: Inserção de erro no módulo 1

No diagrama de tempo apresentado na figura 3.9, cada bit de um vetor de temperatura é sintetizado como um flip-flop-D (ver seção de resultados). O vetor `T_in` representa os valores de temperatura corretos de entrada nos módulos 2 e 3 de registradores de entrada, que nesse caso oscila entre 1^o C e $1,5^o$ C. O vetor `T_in_erro` representa a inserção de uma falha do tipo *stuck-at-1* (curto-circuito entre a saída do flip-flop-D e +Vcc) no bit-9 (`T_in_erro(9)`) e uma saída de flip-flop-D flutuando em `T_in_erro(10)` representado pelo `L` (*weak logic low*).

Observa-se que a saída `T_out` é sempre correta seguindo a entrada `T_in` à cada subida de *clock* mesmo havendo os erros injetados.

A palavra de erro confirma o erro no módulo 1 assumindo o valor de e=110.

Caso 2: Falha em circuito aritmético

(Etapa: `conversor`)

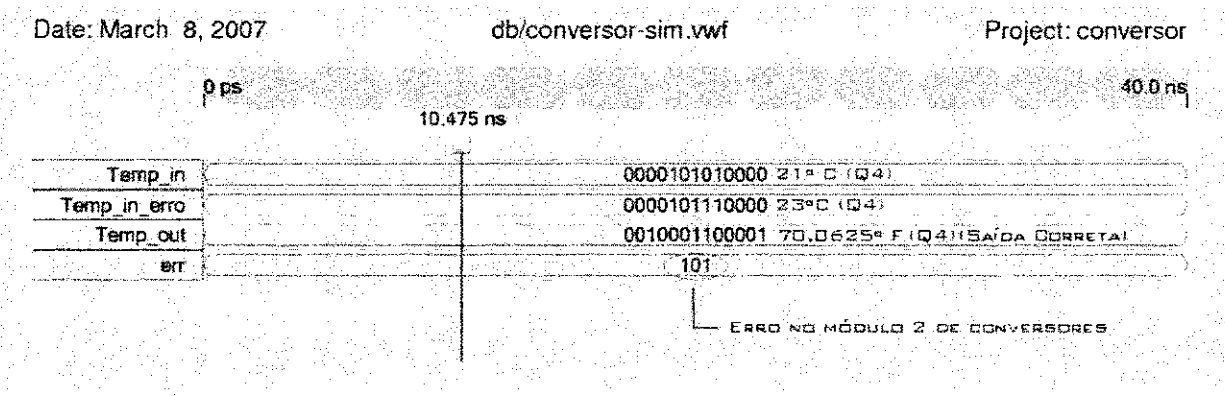

Figura 3.10: Inserção de erro em circuito aritmético do conversor

A Figura 3.10 apresenta os resultados da simulação para o conversor de escala de temperatura. A entrada correta do sensor representada em ponto fixo na base 4 (Q4) é de 21^o C. Um falha é inserida no módulo 2 de conversores através de uma entrada errada, o que produziria uma conversão diferente da dos módulos 1 e 3 simulando uma falha no circuito aritmético. Observa-se que 21^o C corresponde à 69.8^o F. Logo, a saída `T_out`, mesmo na presença da falha, está correta visto que seu valor é $70,0625^o$ C, sendo o erro decorrente do processo de aproximação da expressão de conversão (seção 3.1.2). A palavra de erro e=101 confirma que o módulo 2 é o responsável pela falha.

As falhas na etapa do filtro FIR (`filtro_da_media`) não são apresentadas pelo fato de que do ponto de vista modular, tal etapa representa circuitos aritméticos sendo os resultados apresentados para o conversor também válidos na filtragem.

Caso 3: Falha em bits de *flag*

(Etapa: `memoria_fifo` em operação síncrona)

Foi considerado aqui a operação síncrona da fila, já que dessa forma não deve-se ter a ativação dos flags *full* e *empty* e possa-se assim inserir erros para avaliação da funcionalidade da técnica.

Nos diagramas de tempo apresentados na Figura 3.11 são inseridos erros de *flag* de *full* no módulo 1 e *empty* no módulo 3 para entradas de temperatura que oscilam em torno da temperatura ambiente.

O vetor de dados de leitura `rd_data` recebe os valores da pilha corretamente de acordo com sua freqüência de clock, mesmo havendo módulos lançando *flags* errados.

A palavra de erro apresenta corretamente o diagnóstico já que, não havendo erro seu valor deve ser $err = (1FF)_{16} = (111111111)_2$ e observa-se que no instante de erro no flag *empty* o

vetor de erros assume o valor $err = (0FF)_{16} = (011111111)_2$ que é exatamente a palavra de erro de flag *empty* no módulo 3. O mesmo acontece quando há erro de flag *full* no módulo 1 $err = (1F7)_{16} = (111110111)_2$ e por fim para o erro em ambos tem-se $err = (0F7)_{16} = (011110111)_2$

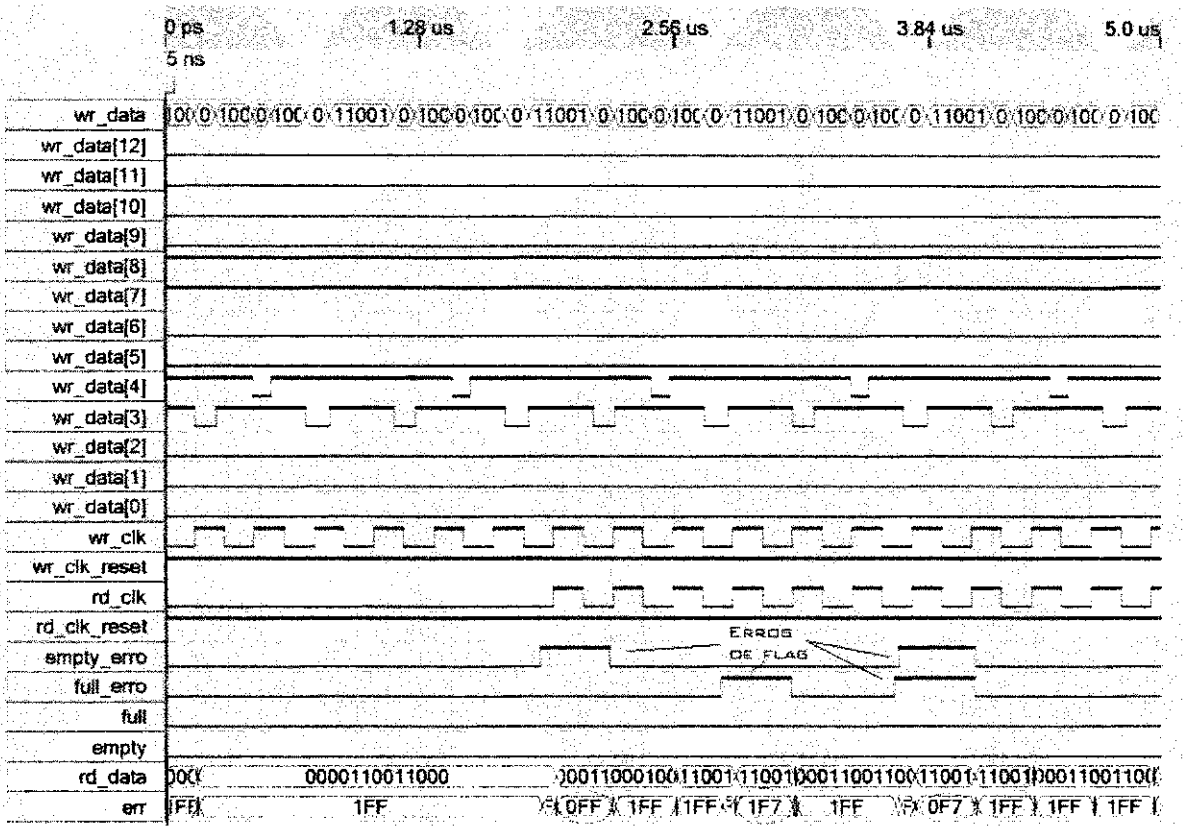

Figura 3.11: Inserção de erros de flag

3.4.2 Testes: Códigos de Hamming

No caso do código de hamming implementado, já se é previamente sabido que a capacidade de correção de erros é de somente 1 erro. Para verificação da técnica fez-se a inserção de erros nos módulos de `registrador_de_entrada`, `conversor` e `filtro_da_media`.

Apresentaremos aqui somente o caso de erro no módulo de registrador de entrada. Os testes de inserção de falha nos outros módulos produziram resultados semelhantes já que a operação do codificador e descodificador se dá da mesma forma.

Falha tipo *stuck-at1*

(Etapa: `registrador_de_entrada`)

A Figura 3.12 ilustra a situação de uma temperatura de entrada de 25° C. O vetor de dados `OUT_ENCODER` representa a saída do codificador (18 bits). Nessa palavra código, é feita a inversão do bit 6 (`OUT_ERROR_BLOCK(6)`) representando um curto circuito da saída do flip-flop-D com +Vcc (stuck-at1). O código corretor de erro implementado no descodificador garante

uma saída correta (OUT_DECODER = 25°), fazendo com que a temperatura final carregada no microprocessador (TEMP_F) de 77,25° F.

Nota-se também que nos instantes inicias há uma oscilação na temperatura de saída TEMP_F devido a operação do filtro da média que nos primeiros ciclos de clock não possui entradas passadas válidas. Vê-se que a partir de 5,12 μs o flag de full passa a ser "setado"periodicamente de forma correta já que a taxa de saída de dados da fila é a metade da taxa de entrada.

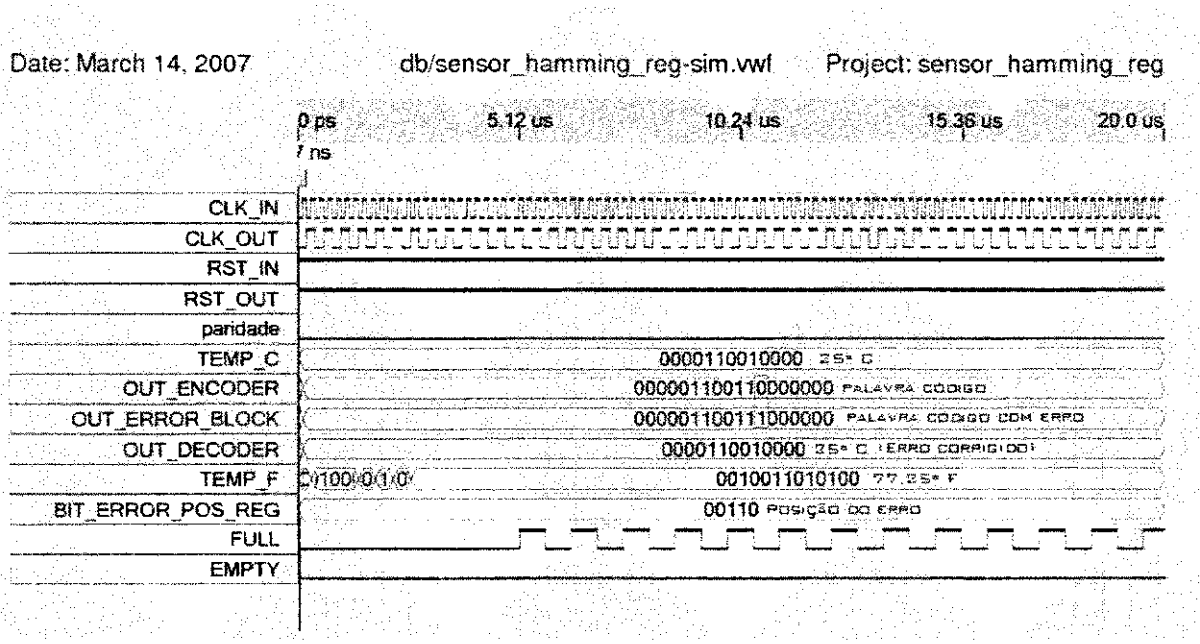

Figura 3.12: Operação do sistema com abordagem de código de hamming

Capítulo 4

Análise comparativa, resultados e conclusões

4.1 Síntese dos circuitos

Pode-se dividir basicamente em 3 passos um projeto que emprega linguagem de descrição de circuitos.

- Descrição VHDL, Verilog HDL, SystemC, etc.
- Compilação e simulação.
- Síntese: rede de ligações, posicionamento, estruturas disponíveis na tecnologia.

A ferramenta de síntese utilizada foi a QUARTUS II/ALTERA V4.0 para que fossem gerados os circuitos que serão definidos no FPGA. As seções que seguem apresentam os circuitos sintetizados das técnicas de tolerância a falha abordadas neste trabalho.

Sistema original - *Top-Level*

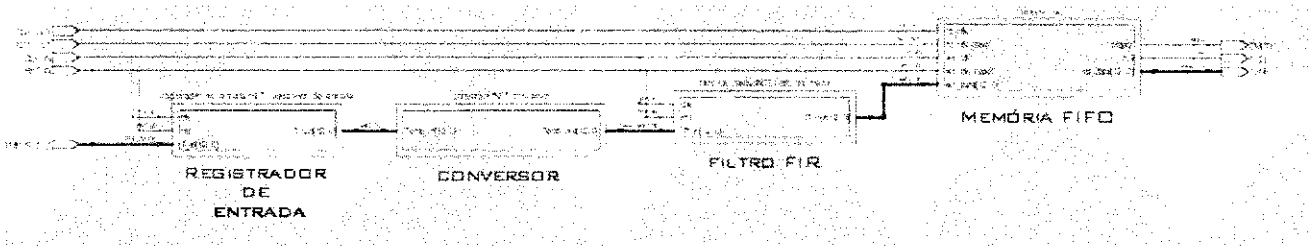

Figura 4.1: *Top-Level* Nível RTL do sistema (Quartus II 4.0)

4.1.1 Síntese para abordagem TMR

Devido à quantidade de circuitos sintetizados ser bastante grande, não é viável apresentar níveis mais baixos de síntese de todos os blocos. Como exemplo, mostraremos aqui somente a implementação de mais baixo nível do bloco conversor.

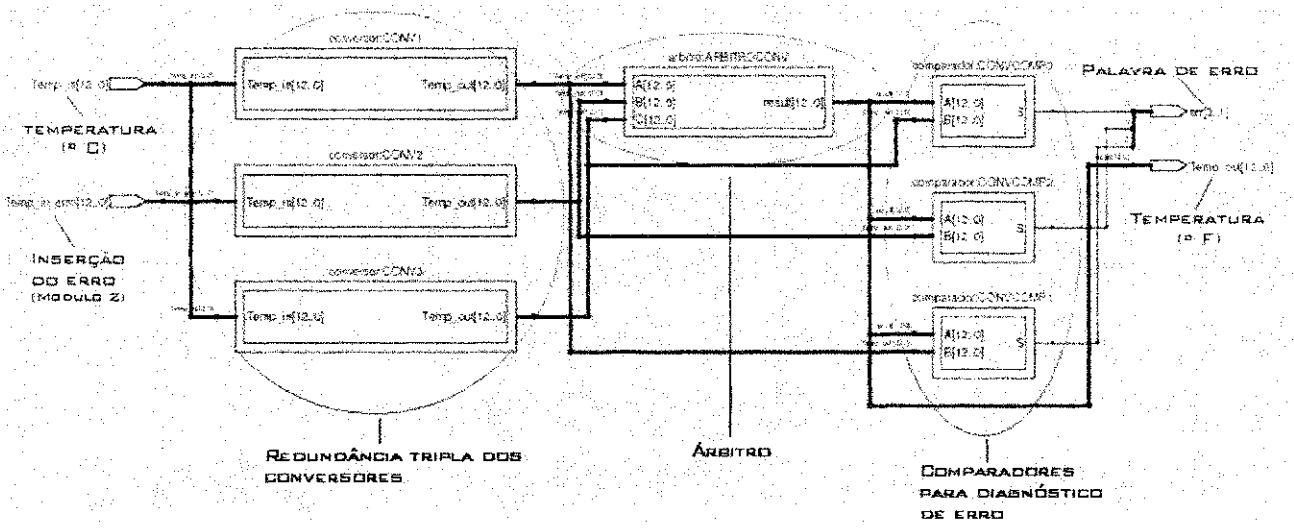

Figura 4.2: TMR na etapa de conversores - (Quartus II 4.0)

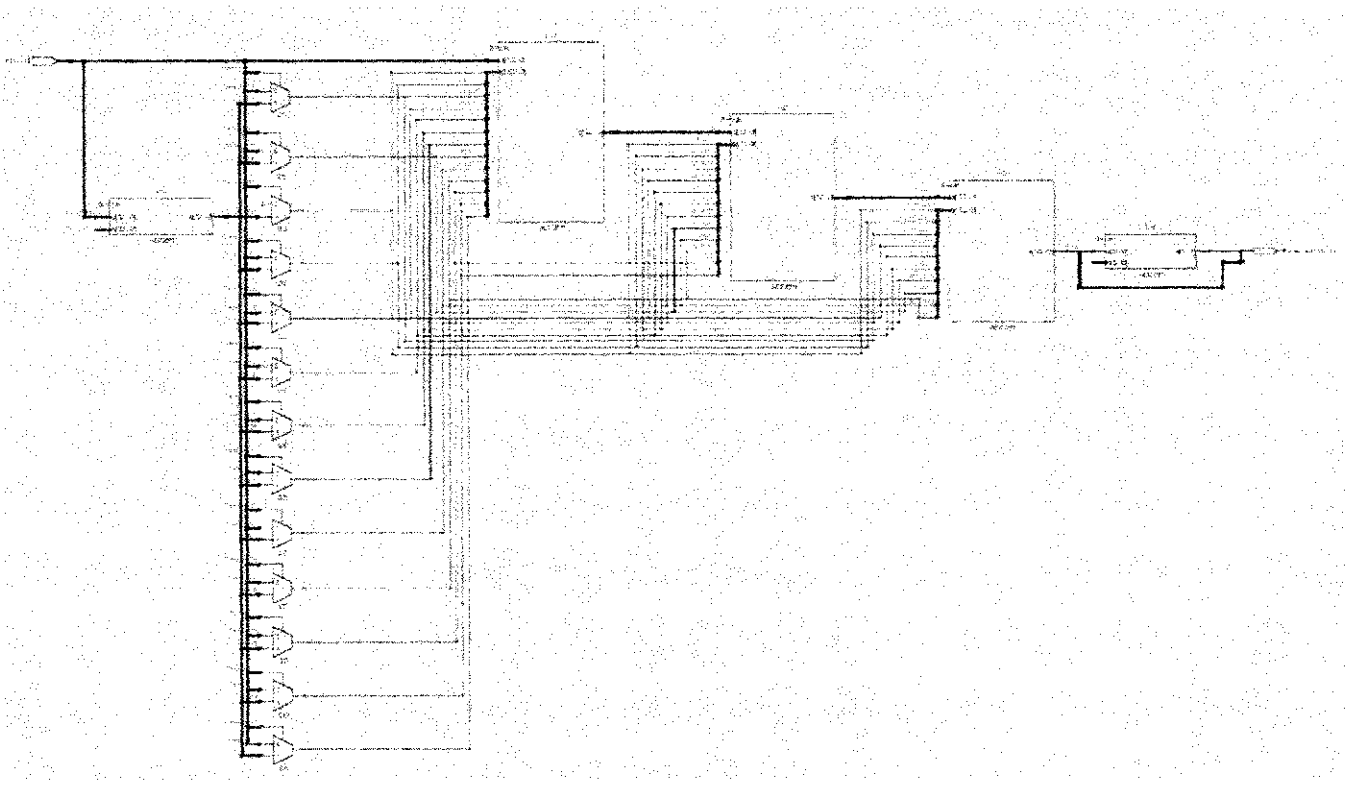

Figura 4.3: Síntese de um conversor de temperatura em ponto fixo 13 bits (Q4)

4.1.2 Síntese para a abordagem de códigos de hamming

A abordagem de hamming difere do sistema na sua forma original somente pela introdução dos blocos `encoder` e `decoder`. As Figuras 4.4 e 4.5 mostram o nível RTL sintetizado.

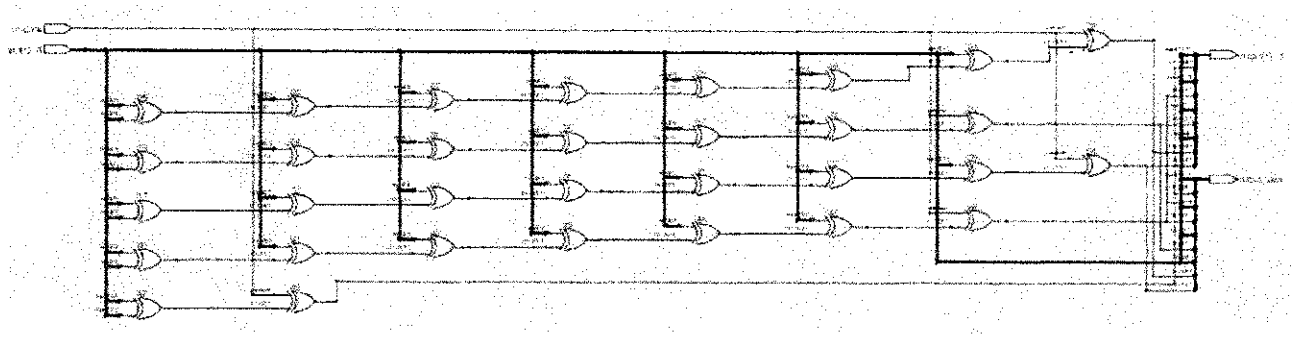

Figura 4.4: Nível RTL do codificador hamming

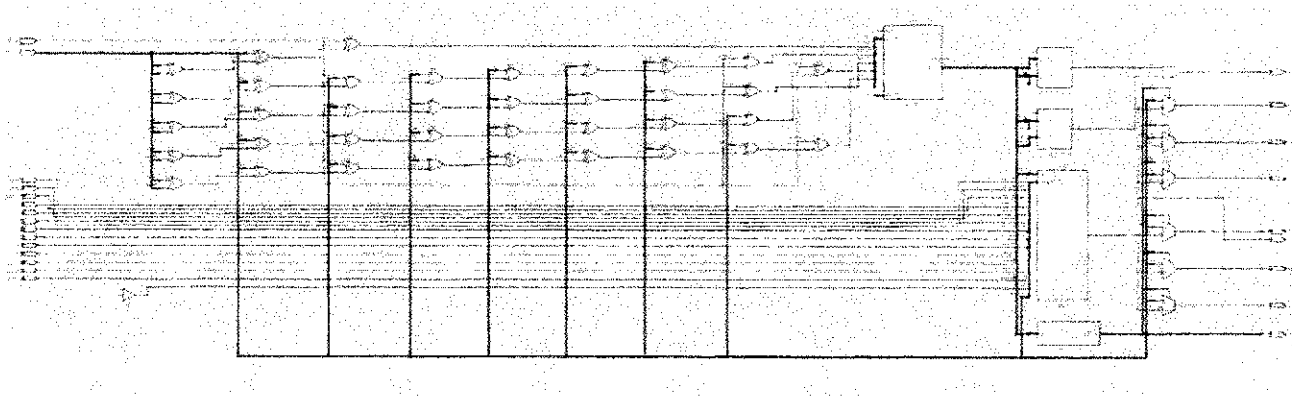

Figura 4.5: Nível RTL do descodificador hamming

Os parâmetros utilizados na comparação entre as duas técnicas buscam representar os custos em potência e em área. As tabelas apresentadas abaixo comparam tais grandezas entre os sistema original, o sistema com uso da TMR, com uso de códigos de hamming e uma combinação das duas técnicas. Tais resultados foram obtidos através da ferramenta de síntese Quartus II v4.0 e as estimativas de potência foram feitas com a funcionalidade *Simulation Based Power Estimation* presente também na ferramenta. Os valores apresentados nas tabelas a seguir são representativos para o FPGA Altera, série Cyclone EP1C20F400C6.

	Sistema simples	TMR	Hamming	TMR/Hamming
Registradores	106	392	106	390
Pinos	32	52	48	52
Registradores (LC's)	61	307	61	305
LUT's por Registrador (LC)	45	85	46	83
Nº de LC's (registradores de entrada)	13	135	87	135
Nº de LC's (conversor)	64	141	125	141
Nº de LC's (filtro FIR)	173	435	244	435
Nº de LC's (memória FIFO)	774	2349	831	831
Nº de LC's total	1024	3060	1287	1542
Nº de bits M/C [1] (reg. de entrada)	0	até 13	1	até 13
Nº de bits M/C (conversor)	0	até 13	1	até 13
Nº de bits M/C (filtro)	0	até 13	1	até 13
Nº de bits M/C (memória FIFO)	0	até 15	3	3

Tabela 4.1: Comparação entre as técnicas implementadas - Uso de células lógicas e capacidade de mascaramento/correção de bits (Quartus II 4.0)

[1] **LC's** = ***Logical Cells***

[2] **M/C** = **M**ascaráveis ou **C**orrigíveis. No caso da TMR faz-se necessário que tal número de erros ocorra no mesmo módulo.

A Tabela 4.2 apresenta uma comparação entre o consumo de potência considerando as arquiteturas implementadas. Tanto para o sistema simples, para o sistema com a TMR e com uso de códigos de hamming o arquivo que contém os estímulos (*testebench file*) foram os mesmos como também o tempo de simulação foi igual para os quatro casos.

	Sistema simples	TMR	Hamming	TMR/Hamming
Potência interna	105 mW	105 mW	105 mW	105 mW
Potência dos elementos lógicos	120,14 mW	166,16 mW	201,86 mW	184,12 mW
Potência de clock	14,84 mW	51,08 mW	15,01 mW	22,41 mW
Potência total interna	239,98 mW	322,24 mW	321,87 mW	322,03 mW
Potência total I/O	2,41 mW	21,07 mW	3,8 mW	18,4 mW
Potência total	242,40 mW	343,31 mW	325,67 mW	340,43 mW

Tabela 4.2: Comparação entre as técnicas implementadas - Estimativa de consumo de potência (Quartus II 4.0)

4.2 Conclusões

Neste trabalho foram apresentadas implementações de duas técnicas básicas de tolerância a falha utilizando a linguagem de descrição de hardware VHDL. A importância do assunto aqui explorado fica evidente ao observar-se que a área de tratamento de defeitos, falhas e erros é sem dúvida fundamental para as nanotecnologias emergentes em que o projeto de circuitos tolerantes estará sempre presente.

Os dados obtidos para os três diferentes esquemas de tolerância a falha, apresentados nas Tabelas 4.1 e 4.2, evidenciam o maior *overhead* de área quando é utilizada a técnica de redundância modular tripla - TMR. A área extra necessária é de aproximadamente 298 % em relação ao sistema simples (desse alto valor de 298% obtido para a TMR deve-se descontar o percentual de células lógicas extras devido aos circuitos de diagnóstico que exigem também um número elevado de registradores. A importância desses circuitos está no fato de que através do diagnóstico pode-se mapear e isolar a área defeituosa utilizando um processo de reconfiguração, por exemplo), enquanto que com uso de códigos de Hamming esse *overhead* não atingiu 25%. No entanto, a abordagem TMR é claramente superior quando considera-se a habilidade de mascarar/corrigir erros. A extensão da TMR para NMR com utilização de 5,7,9 ou mais blocos é uma opção quando se deseja aumentar mais ainda a confiabilidade, porém, esse aumento é feito às custas de maior conteúdo de área extra.

Os melhores resultados foram obtidos com a abordagem complementar. O critério adotado para a utilização conjunta das duas técnicas é baseado em uma análise preliminar afim de identificar o(s) módulo(s) que contribuem mais significativamente para o custo em área do projeto, e redefinir um esquema de tolerância a falha para tal módulo reduzindo assim o custo em área total. Note que, para o estudo de caso apresentado, o módulo da memória FIFO é responsável por quase 75 % da área do projeto. Modificando para o uso de códigos de Hamming nesse estágio e mantendo a solução TMR no restante do sistema, a área extra requerida cai expressivamente para 48 % e o consumo de potência correspondente é levemente alterado.

Fica evidente com esse estudo de caso que a escolha da técnica a ser utilizada em um determinado projeto tem uma forte dependência com a arquitetura original do sistema e essa escolha não pode ser feita baseada somente em simples análises teóricas [7]. Como foi mostrado, a arquitetura primitiva do sistema pode fornecer informação útil para a escolha do melhor método de tolerância a falha.

Vale ressaltar que não só a TMR e a codificação de hamming descritas constituíram estratégias de tolerância a falha, mas também as próprias propostas feitas no sistema original contribuem para a redução de erros de uma forma geral como a questão da comparação dos ponteiros na memória FIFO [14] e a redução da complexidade nas operações com a abordagem de ponto fixo.

O caráter do estudo desenvolvido envolvendo descrição de hardware, eletrônica CMOS e circuitos com tolerância a falha é de estimular as pesquisas nessa área que é fundamental para o futuro da eletrônica.

Como sugestão de trabalhos futuros, pode-se avaliar a funcionalidade de outras técnicas que compreendam códigos corretores de erros mais sofisticados, aplicar as técnicas a outros contextos e avaliar o rendimento das mesmas sob outros pontos de vista não abordados neste trabalho.

Referências Bibliográficas

[1] J. Han and P. Jonker, "A system architecture solution for unreliable nanoelectronic devices," *IEEE transactions on nanotechnology*, vol. 1, no. 4, 2002.

[2] D. T. Franco, J.-F. Naviner, and L. Naviner, "Yield and reliability issues in nanoelectronic technologies," *ANN. Télécommun.*, vol. 61, no. 11 - 12, 2006.

[3] K. Chakraborty and P. Mazumder., "Reliability and fault tolerance of rams," 2002.

[4] J. M. M. Smith, *Application Specified Integrated Circuits.* Addison Wisley, 2004.

[5] T. Weber, "Apostila: Fundamentos de tolerância à falha." 2004.

[6] J. Segura and C. F. Hawkins, *CMOS electronics: How it works, How it fails.* John Wiley and Sons, INC, 2004.

[7] M. F. K. Nikolic, A. Sadek, "Architectures for reliable computing with unreliable nanodevices," *IEEE-NANO*, vol. 61, 2001.

[8] T. M. Michael Spica, "Do we need anything more than single bit error correction (ecc)?," *International Workshop on Memory Technology, Design and Testing*, 2004.

[9] P. J. Fox, *Low cost fault tolerant by I/O inversion.* PhD thesis, Jesus College, maio 2006.

[10] J. Han, *Fault-tolerant Architectures for Nanoelectronic and Quantum Devices.* PhD thesis, Tsinghua University, agosto.

[11] Institute of Electrical and Electronics Enginee, *IEEE Standard Vhdl Language Reference Manual: IEEE Std 1076-1993.*

[12] National Semiconductor Corporation, *LM73 - 2.7V, SOT-23, 11-to-14 Bit Digital Temperature Sensor with 2-Wire Interface*, Agosto 2006.

[13] R. d'Amore, *VHDL - Descrição e síntese de circuitos.* Rio de Janeiro: LTC, 2005.

[14] M. Berg, "Presentation: Metodologies for reliable design implementation." 2004 MAPLD International Conference, Setembro 2004.

[15] "www.wikipedia.org/wiki/codedehamming. acessado em 11/03/2007."

[16] Altera manual, *Using ModelSim-Altera in a Quartus II Design Flow*, 1.2 ed., Dezembro 2002.