UNIVERSIDADE FEDERAL DE ITAJUBÁ

IMPLEMENTAÇÃO DE UM MEDIDOR DE TAXA DE ERRO DE BITS COM CRC UTILIZANDO LINGUAGEM VHDL

Kelly de Pinho Miranda

ITAJUBÁ – MG
2003

UNIVERSIDADE FEDERAL DE ITAJUBÁ

Pró-Diretoria de Pesquisa e Pós-Graduação

Programa de Pós-Graduação em Engenharia Elétrica

IMPLEMENTAÇÃO DE UM MEDIDOR DE TAXA DE ERRO DE BITS COM CRC UTILIZANDO LINGUAGEM VHDL

Kelly de Pinho Miranda

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como requisito parcial à obtenção do título de **Mestre em Ciências em Engenharia Elétrica**

Orientador: Prof. Germano Lambert Torres, Dr.
Co-Orientador: Prof. Francisco Martins Portelinha, MS.

Itajubá, outubro de 2003

Dedicatória

Aos meus pais, Paulo Mateus de Miranda e Eunelália Maria de Pinho Miranda que sempre se fizeram presentes ao meu lado em todas as decisões da minha vida, pela formação de caráter a mim ensinados, sempre aconselhando e incentivando.

Aos meus avós, José Manoel de Miranda e Benedita Maria de Miranda, que sempre me acolheram e deram o melhor de si.

Ao meu namorado, Carlos Garcia Gusman, e à minha amiga Juliana Lopes dos Santos que mesmo distantes, sempre me apoiaram.

Agradecimentos

Agradeço a *Deus* por mais essa conquista. Sua presença me confortou nas horas difíceis e sempre me transmitiu a força necessária para continuar.

Ao Prof. Francisco Martins Portelinha, M.S.c, pela dedicação e orientação durante a elaboração deste trabalho.

Ao meu orientador, Prof. Germano Lambert Torres, Dr.

Ao Prof. Robson Luiz Moreno, Dr., pela colaboração no desenvolvimento deste trabalho.

Resumo

Este trabalho apresenta o projeto de um dispositivo de baixo custo para medir a taxa de erro de bits nos sistemas de comunicação de dados. Através da detecção dos erros ocorridos durante a transmissão dos dados em um certo intervalo de tempo, obtém-se a medida de taxa de erro de bits ("BER meter"). Este trabalho mostra também o programa desenvolvido em linguagem VHDL utilizando a ferramenta MAX+PLUS II, (versão 10.2, da Companhia ALTERA) que simula o funcionamento do dispositivo de medida de taxa de erro de bits proposto.

Abstract

This work presents the project of a device with a low cost to measure the bit error rate in data communications systems. Through the detention of the errors occurred during the transmission of the data in a certain interval of time, it is gotten measure it of bits error rate. This work also shows the program developed in VHDL language by using MAX+PLUS II tool, (version 10.2, of ALTERA Company) that simulates the functioning of measurer device bit error rate proposed.

ÍNDICE

Índice de Figuras

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 5

Índice de Equações

Capítulo 2

Capítulo 3

Capítulo 4

Tabela de Notações

AM – Modulação por Amplitude

ASK – Amplitude Shift Keying

BER – Bit Error Rate

BSC – Binary Symetric Channel

CPLD – Complex Programmable Logic Devices

CRC – Códigos de Redundância Cíclica

ECL – Emitter Coupled Logic

FM – Modulação por Freqüência

FPGA – Field Programmable Gate Array

FSK – Frequency Shift Keying

LCD – Display de Cristal Líquido

PM – Modulação por Fase

PSK – Phase Shift Keying

TTL – Transistor Transistor Logic

VHDL – Very High Speed Integrated Circuit Hardware Description Language

CAPÍTULO 1 – INTRODUÇÃO

A necessidade de se transmitir dados de um ponto a outro com precisão, é quesito principal nos sistemas de comunicações. Para que isso ocorra, é necessário que o canal onde os dados trafegam possua as características necessárias para que não ocorram erros durante a transmissão dos dados. Por essa razão, a avaliação em tempo real de um sistema de comunicação de dados é de grande importância.

Este trabalho apresenta o desenvolvimento de um projeto capaz de detectar os erros ocorridos durante a transmissão de dados em um dado intervalo de tempo, obtendo assim, uma *Medida de Taxa de Erro de Bits* (*"BER meter"*). Este trabalho desenvolve o projeto programável de códigos CRC de até 32 bits. Para a implementação deste projeto utilizou-se CRC (Códigos de Redundância Cíclica), pois são códigos de alta eficiência na detecção de erros durante a transmissão dos dados [1]. Através da detecção destes erros, e da medida de BER, pode-se fazer uma avaliação do sistema onde os dados estão sendo transmitidos. O projeto do circuito foi desenvolvido em linguagem de programação de hardware de alto nível VHDL. Escolheu-se esta linguagem de programação devido à possibilidade de gravar em dispositivos lógicos programáveis.

Apresentação do Trabalho

No capítulo 2 é descrito um sistema de comunicação de dados e seus blocos internos: codificador de fonte; modulador; canal; demodulador e decodificador. São definidos alguns critérios de desempenho e medidas de desempenho nos sistemas de comunicação. É descrito como se avalia o desempenho do meio físico em sistemas de comunicação e como os códigos de redundância cíclica são úteis na detecção de erros na transmissão digital.

No capítulo 3 é apresentada a teoria matemática de códigos de redundância cíclica - CRC que é utilizado no projeto do medidor de taxa de erro de bits. Um sistema global de medida de taxa de erro é ilustrado, contendo a descrição de como os bits de erros são detectados.

No capítulo 4 é apresentado o projeto do dispositivo lógico medidor de taxa de erro de bits (BER meter) utilizando linguagem VHDL. Este capítulo apresenta o desenvolvimento do projeto em três blocos: gerador de dados; decodificador e medidor de BER. São apresentadas simulações feitas em cada bloco separadamente e o projeto total incluindo simulações feitas em software.

No capítulo 5 são apresentadas as conclusões sobre o projeto e algumas sugestões de trabalhos futuros para otimização dos sistemas de comunicações digitais.

CAPÍTULO 2 – AVALIAÇÃO DE DESEMPENHO EM SISTEMAS DE COMUNICAÇÕES DIGITAIS

2.1 - Introdução

Este capítulo apresenta os parâmetros teóricos e práticos para avaliação de desempenho dos sistemas de comunicações. Apresenta também a configuração que deve ser utilizada para levantamento da taxa de erros e uma introdução de códigos de redundância cíclica.

A medição de taxa de erro de bits utiliza técnicas de detecção de erros em uma seqüência digital. Esta seqüência é originada por um gerador pseudo-aleatório no transmissor. Isto permite medir o desempenho de um sistema de transmissão digital. Vamos descrever a técnica de detecção de erro utilizada. Esta técnica pode ser utilizada para otimizar o sistema de comunicação de dados. A técnica é simples e baseada em uma comparação bit a bit entre a seqüência recebida e a seqüência de referência gerada localmente.

2.2 – Elementos de um Sistema de Comunicação Digital

A figura 2.1 ilustra o diagrama de blocos de um sistema de comunicação digital que é composto dos seguintes blocos:

Codificador de Fonte, Modulador, Canal, Demodulador e *Decodificador* [2].

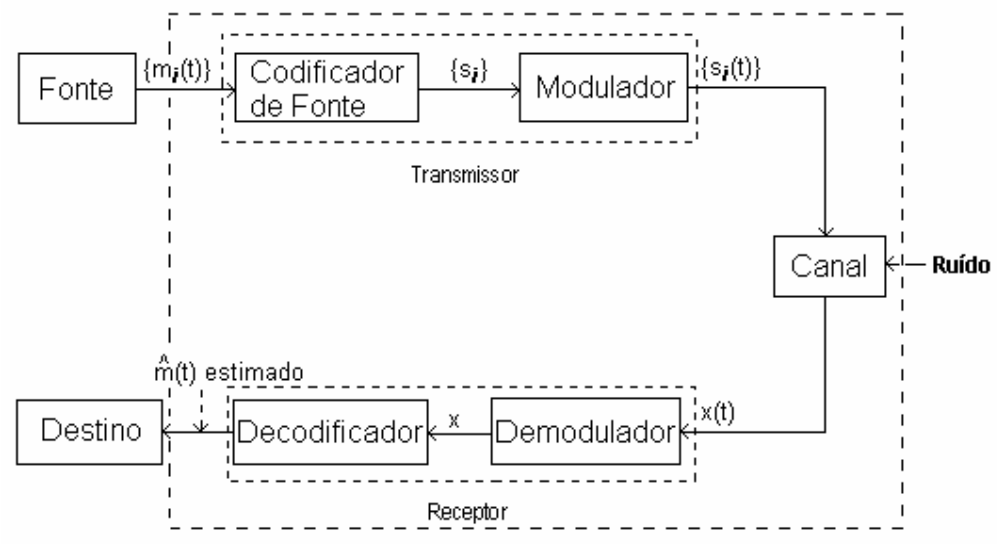


Figura 2.1 – Diagrama de blocos de um sistema de comunicação digital

2.2.1 – Codificador de Fonte

O *codificador de fonte* recebe a informação emitida pela fonte e se necessário transforma esta informação em seqüência de dígitos binários chamado de seqüência de informação [3]. No caso de uma fonte contínua, envolve conversão *analógico-para-digital* (A/D). O codificador de fonte é idealmente designado tal que: o número de bits por unidade de tempo necessário para representar a saída da fonte seja minimizado, e a saída da fonte no lado do receptor possa ser uma reconstrução da seqüência de informação sem que haja ambigüidades.

2.2.2 – Modulador

O modulador transforma o sinal s_i de duração de T segundos proveniente do codificador de forma a adequá-lo ao canal de transmissão [3]. Modulação é definida [2] como a técnica que permite a alteração de uma ou mais características (amplitude, fase, freqüência) de um sinal de onda (denominado portadora) em função das características de um outro sinal (chamado sinal modulador ou modulante). No caso de comunicação de dados, trata-se de um sinal binário. Dependendo da característica a ser alterada da portadora podemos ter modulação por amplitude - AM (alteração da amplitude da portadora), modulação por freqüência - FM (alteração da freqüência da portadora), modulação por fase – PM (alteração da fase da portadora), modulação combinada de amplitude e fase - QAM (alteração da amplitude de duas portadoras que são defasadas e somadas - amplitude + fase), etc.

Para o sinal modulante digital, as três principais técnicas de modulação AM, FM e PM passam a ser denominadas ASK (Amplitude Shift Keying), FSK (Frequency Shift Keying) e PSK (Phase Shift Keying) respectivamente [2].

2.2.3 – Canal

A forma de onda $s_i(t)$ da figura 2.1 entra no *canal* e é corrompida pelo *ruído* como é definida por Shu e Costello [3]. Existem vários tipos de canais de transmissão. São eles: linha telefônica (par trançado), cabo coaxial, fibra óptica, rádio freqüência, microondas, satélite e outros tipos de canais de transmissão.

Os meios de transmissão (canal) diferem com relação à banda passante, potencial para conexão ponto a ponto ou multiponto, limitação geográfica devido à atenuação característica do meio, imunidade a ruído, custo, disponibilidade de componentes e confiabilidade [4].

A escolha do meio de transmissão adequado às aplicações é importante não só pelos motivos mencionados, mas também pelo fato de que ele influencia diretamente no custo das interfaces com a rede.

A tabela 2.1 define alguns dos meios de transmissão existentes [4].

Meios de Transmissão

Nome	Par Trançado
Definição	Dois fios enrolados em espiral para reduzir o ruído e manter constantes as propriedades elétricas através de todo o seu comprimento.
Características	Baixo custo; não permite a operação em altas freqüências (> 100 Khz); baixa imunidade ao ruído, principalmente em freqüências altas; a capacidade de transmissão depende, principalmente, da impedância e atenuação do par, que dependem do diâmetro e do comprimento do fio.
Considerações	A transmissão pode ser analógica ou digital. Banda passante é alta, considerando o fato de ele ter sido projetado para o tráfego analógico telefônico. Pode chegar até várias dezenas de metros com taxas de transmissão de alguns megabits por segundo. É normalmente utilizado na transmissão digital de sinais. Outra aplicação típica é a ligação ponto a ponto entre terminais e computadores e entre estações da rede e o meio de transmissão.
Aplicações	Interligação de sistemas distantes, através da Internet; interligação de "terminais" em Intranet.

Nome	Cabo Coaxial
Definição	Constituído de um condutor interno circundado por um condutor externo, tendo, entre os condutores, um dielétrico que os separa. O condutor externo é circundado por outra camada isolante.
Características	Custo um pouco superior ao "par trançado"; maior imunidade a ruídos, em relação ao par trançado; distância, limitada em função da atenuação (centenas de metros); opera em freqüências altas, o que permite a transmissão em taxas que podem chegar a 20 Mbps; a transmissão em banda larga fornece uma imunidade ao ruído melhor do que em banda base.
Considerações	Existe uma grande variedade de cabos coaxiais, cada um com características específicas. Alguns são melhores para transmissão em alta freqüência, outros têm atenuação mais baixa, outros são mais imunes a ruídos e interferências, etc. O cabo coaxial, ao contrário do par trançado, mantém uma capacitância constante e baixa, teoricamente independente do comprimento do cabo.
Aplicações	Redes locais (LAN) em: - Banda base (50W) – transmissão digital (rede internet). - Banda larga (75W) – portadora modulada com a informação (TV a cabo)

Nome	Fibra Óptica
Definição	O cabo óptico consiste em um filamento de sílica ou plástico, por onde é feita a transmissão da luz. Ao redor do filamento existem outras substâncias de menor índice de refração, que fazem com que os raios sejam refletidos internamente, minimizando assim as perdas na transmissão.
Características	Alta imunidade a ruídos; custo do sistema superior ao dos cabos coaxiais; dificuldade de instalação e manutenção dos sistemas, principalmente nas conexões e conversões elétricas.
Considerações	A composição da fibra é determinante na sua atenuação, que pode ser causada pela dispersão ou absorção da luz por elementos presentes no núcleo.
Aplicações	Em linhas de longa distância utilizadas pelas companhias telefônicas, com distâncias próximas de 50 Km sem a necessidade de repetidores; utilizadas em sistemas com taxas de transmissão que chegam a 150 e a 620 Mbps numa única fibra unidirecional.

Nome	Sistemas Rádio: Redes Sem Fio
Definição	Transmissão "através do ar", em canais de freqüência de rádio ou infravermelhos.
Características	Permitem altas taxas de transmissão; flexibilidade de localização.
Aplicações	Transmissões que exigem altas taxas de transferência de informação e transmissão que necessita alcançar regiões remotas.

Nome	Sistema de Comunicação por Satélite
Definição	É um rádio repetidor no espaço (transponder). Um sistema de satélite consiste de um transponder, uma estação terrestre para controlar sua operação, e uma rede de estações terrestres que gerenciam o trânsito na transmissão e recepção de dados nos satélites.
Características	Permite alta capacidade de transmissão; alta qualidade e confiabilidade durante a transmissão; operação em múltiplo acesso; facilidade para alcançar regiões remotas; flexibilidade para expansão (estação terminais).
Aplicações	Transmissão que exige confiabilidade, alta qualidade, alta capacidade e transmissão que necessita alcançar regiões remotas.

Tabela 2.1 – Meios de Transmissão

2.2.4 – Demodulador

O *demodulador* processa cada forma de onda recebida de duração de T segundos e produz uma saída que seja discreta (quantizada) [2]. A seqüência na saída do demodulador corresponde à seqüência codificada, ou seja, ele executa o inverso do modulador.

2.2.5 – Decodificador

O *decodificador* transforma a seqüência recebida do demodulador em uma seqüência estimada $\hat{m}(t)$ que é entregue ao destino. A seqüência estimada $\hat{m}(t)$ deve ser uma reprodução fiel da seqüência emitida pela fonte [2].

Em um sistema de comunicação digital, a *fonte* de informação emite símbolos representados na forma binária por 0 ou 1, formando uma seqüência de dados, como ilustrado na figura 2.2:

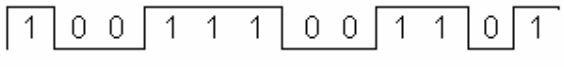


Figura 2.2 – Seqüência de dígitos binários – sinal $m(t)$

O número de bits transmitido por segundo é definido como *Taxa de Bits*.

A inversão de um bit no sinal $\hat{m}(t)$ significa que o sinal $m(t)$ foi corrompido durante a transmissão. Ocorre um erro quando um ou mais bits na seqüência de dados chega trocado no receptor, como ilustrado na figura 2.3:

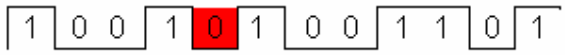


Figura 2.3 – Seqüência de dígitos binários com 1 bit de erro – sinal $\hat{m}(t)$

Poderá ocorrer também a inversão de dois ou mais bits. A inversão deste bit, ou destes bits consiste no erro ocorrido durante a transmissão dos dados através do canal. Neste caso $m_i(t) \neq \hat{m}(t)$. Isto pode ocorrer devido a problemas físicos no canal tal como um surto de tensão que altera um ou mais bits dependendo da duração do mesmo em relação à taxa de bits [2].

A probabilidade de ocorrer 1 bit de erro durante uma transmissão digital de dados é de 75%. A probabilidade de ocorrerem 2 bits de erros consecutivos durante uma transmissão digital de dados é de 15 %. E a probabilidade de ocorrerem 3 bits de erros consecutivos é remota [1]. Ou seja, a probabilidade de ocorrerem erros durante a transmissão digital de dados diminui, quando os erros aumentam consecutivamente.

2.3 – Medida de Desempenho de Sistemas de Comunicação

Ao se interligar dois sistemas para realizar a comunicação de dados, como ilustrado na figura 2.4, normalmente é necessário avaliar o desempenho deste sistema através de parâmetros teóricos e práticos.

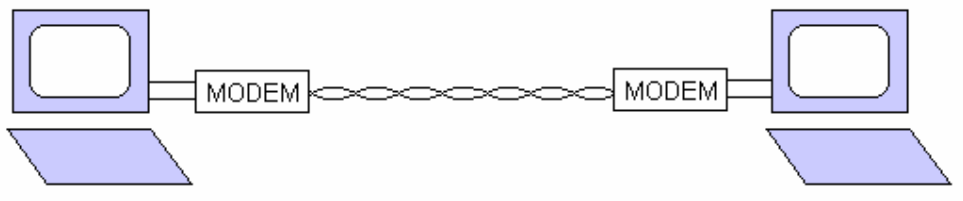


Figura 2.4 – Interligação de dois sistemas de comunicação

O parâmetro teórico utilizado para avaliação do desempenho de um sistema de comunicação é a Probabilidade de Erro *P(e)* e o parâmetro prático é a Taxa de Erro de Bit – BER [1]. Porém, ao se medir o desempenho de um sistema de comunicação na prática, este certamente possuirá uma diferença mesmo que mínima dos valores teóricos medidos. A figura 2.5 [5] ilustra a diferença de valores de desempenho teóricos (curva A) e práticos (curva B) de um modem QPSK:

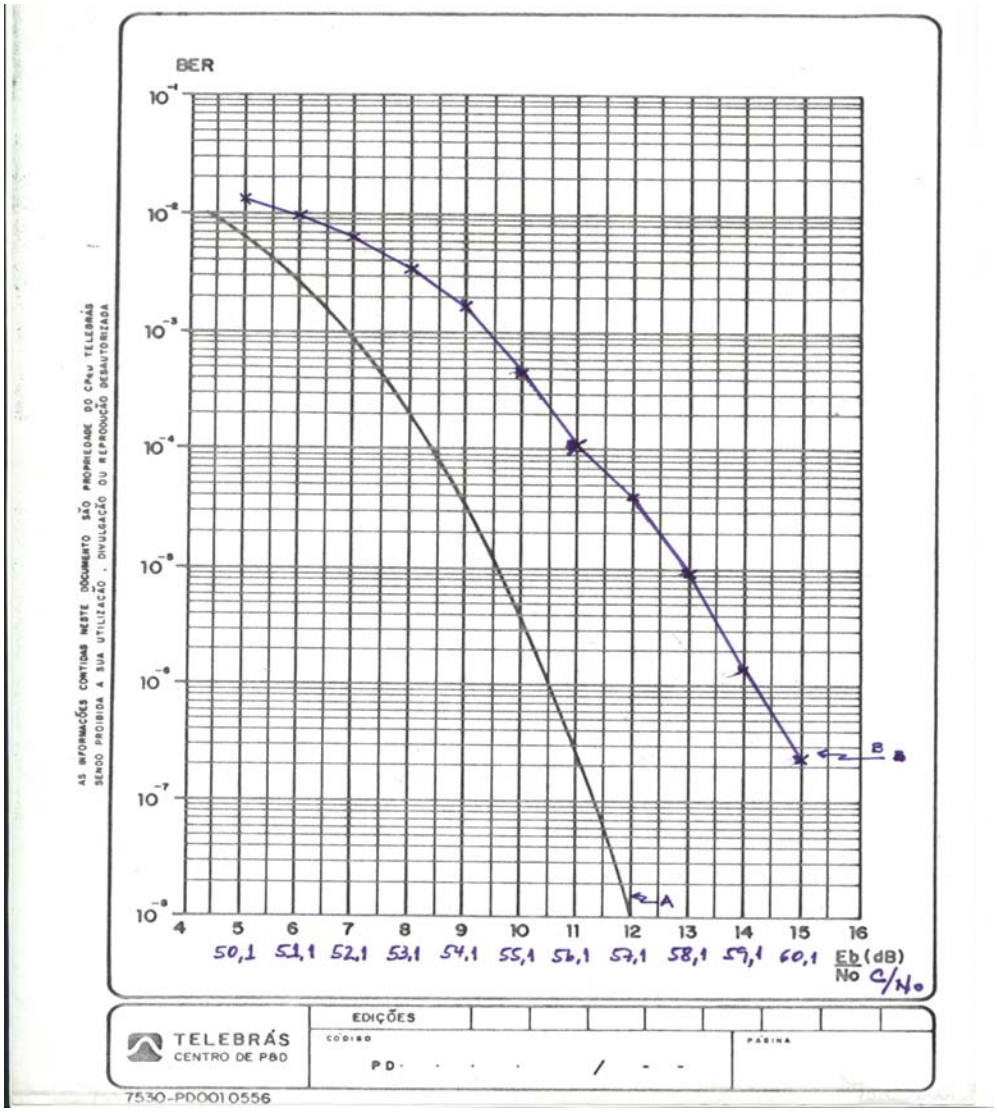


Figura 2.5 - Curvas de Taxas de Erros

A curva “A” traz os valores de desempenho teóricos de um modem QPSK. A curva “B” traz os valores de desempenho práticos (medidos) de um modem QPSK. Embora pequena, pode-se observar que existe uma diferença de valores práticos e teóricos medidos, o que enfoca a importância de se medir o desempenho de um sistema de comunicação em tempo real.

Os parâmetros de desempenho de um sistema devem ser definidos, quando houver a necessidade de medir as características específicas de um dado sistema de comunicação em termos de valores numéricos [6]. Estes parâmetros devem ser definidos para ajustar-se de acordo com o sistema no qual eles serão medidos. É importante que haja uma distinção entre dois tipos de parâmetros que são: os parâmetros orientados ao usuário que mede somente as informações disponíveis entre os pontos finais de um sistema de comunicação digital; e os parâmetros orientados a manutenção e administração que avaliam os componentes individuais do sistema.

O parâmetro mais comum de medida de precisão na transferência de informação dentro de um sistema de comunicação envolve a Taxa de Erro de Bit. Em termos gerais, a *Taxa de Erro de Bit* (BER) é encontrada contando o número de unidades de informação recebidas contendo erros dividido pelo número de unidades de informação recebidas, durante um intervalo de tempo especificado. Desta forma, BER é definido [6] pela seguinte relação:

$$BER = \frac{número\ de\ bits\ errados\ recebidos\ em\ um\ período\ de\ tempo\ definido}{número\ total\ de\ bits\ transmitidos\ sobre\ o\ período\ de\ tempo\ definido} \tag{2.1}$$

Então, defini-se que [6],

$$BER = \frac{N_e}{N_t} = \frac{N_e}{r \cdot t_0} \tag{2.2}$$

onde N_e é o número de bits errados no intervalo de tempo definido; N_t é o número total de bits transmitidos no intervalo de tempo t_0; r é a taxa de bits. Para um processo de medição de dados suficientemente longo, o valor medido estima com precisão do valor da BER [6].

Idealmente, em um sistema de comunicação, o número de unidades de informação transmitidas deveria ser igual ao número de unidades de informação recebidas. Porém, algumas unidades de dados podem ser perdidas ao longo do canal ou podem ser duplicadas (a mesma unidade recebida duas ou mais vezes), ou ainda invertidas e isso resulta em erros na transmissão de informação.

Quando a informação não está estruturada a um caracter, pode-se obter a *Taxa de Erro de Bit* (BER), checando as informações digitais através de erros nas unidades de bits. No

entanto, a informação é, normalmente, estruturada a um caracter (geralmente oito a dez bits de comprimento), ou blocos (até milhões de bits) antes da transmissão. Neste caso, o caracter ou o bloco pode ser tratado como uma unidade, resultando na *Taxa de Erro de Caracter* (CER) ou *Taxa de Erro de Bloco* (BLER) [6]. Na maioria dos sistemas, a presença de um ou mais bits de erros em locais desconhecidos em um caracter ou bloco, faz a unidade inteira ser inutilizada.

A capacidade de avaliar o desempenho de um sistema de comunicação em tempo real é de grande importância para otimizar um sistema de comunicação de dados.

2.4 – Avaliação do Desempenho do Meio Físico em Sistemas de Comunicação

A avaliação de desempenho nos sistemas de comunicação depende do tipo.

Nos sistemas de comunicação analógica, o desempenho é medido por *S/N* (dB), que é a *relação sinal de informação sobre ruído*:

$$\frac{S}{N}(dB) = 10\log\frac{Potência\ \ do\ \ Sinal}{Potência\ \ do\ \ Ruído}(\text{dB}) \qquad (2.3)$$

Por exemplo, em um canal analógico de voz (linha telefônica), a relação sinal-ruído deve ser de no mínimo 30 dB, para que se possa alcançar um grau inteligível de audição.

Em um sistema digital, o desempenho de um canal de comunicação é medido através da *Taxa de Erro de Bit* – BER. Por exemplo, em um canal digital de voz, a taxa de erro de bit deve ser de no mínimo 10^{-5} bits, o que significa 1 bit de erro recebido em 100.000 bits transmitidos.

Em sistemas de comunicações digitais, a *Probabilidade de Erro P*(*e*) e a *Taxa de Erro de Bit* (BER) são parâmetros muito utilizados, embora na prática, tenha uma diferença no significado entre eles. A *P*(*e*) é uma expectativa teórica (matemática) da taxa de erro de bit em um dado sistema de comunicação. BER é uma medida empírica (histórico) do desempenho real do erro de bit de um sistema de comunicação. Por exemplo, se um sistema de comunicação tem uma *P*(*e*) de 10^{-5}, isto significa que matematicamente, pode haver 1 bit de erro em 100.000 bits transmitidos ($1/10^5 = 1/100.000$). Se um sistema de comunicação tem uma BER de 10^{-5}, isto significa que houve 1 bit de erro em 100.000 bits transmitidos. A taxa de erro é medida, e então é comparada com a probabilidade de erro [1]. Desta forma, podemos avaliar o desempenho do sistema de comunicação.

A seguir tem-se uma descrição matemática da equação de Probabilidade de Erro - Pe .

A probabilidade de erro é uma função da relação da potência entre a portadora e o ruído, e o número de possíveis codificações (M-níveis). A relação de potência entre a

portadora e o ruído é a relação da média de potência de portadora pela potência de ruído térmico. A potência da portadora pode ser dada em watts ou dBm, onde:

$$C(dBm) = 10\log\frac{Cwatts}{0.001} \tag{2.4}$$

A potência de ruído térmico é expressa matematicamente como [1]:

$$N = KTB \tag{2.5}$$

onde N é a potência do ruído térmico (W); K é a constante de Boltzmann (1.38×10^{-23} J/K); T é a temperatura em Kelvin = 290 K; B é a largura de faixa (Hz).

Em dBm, tem-se:

$$N(dBm) = 10\log\frac{KTB}{0.001} \tag{2.6}$$

Matematicamente, a relação de potência entre a portadora e o ruído é dado por [1]:

$$\frac{C}{N} = \frac{C}{KTB} \tag{2.7}$$

onde C é a potência da portadora (W); N é a potência do ruído (W).

Em dB, tem-se:

$$\frac{C}{N}(dB) = 10\log\frac{C}{N} = C(dBm) - N(dBm) \tag{2.8}$$

A energia por bit E_b é simplesmente a energia de um único bit de informação. Matematicamente, a energia por bit é expressa como:

$$E_b = CT_b \quad \text{(J/bit)} \tag{2.9}$$

onde E_b é a energia de um bit (J/bit); T_b é a duração de um bit (seg); C é a potência da portadora (W).

Em dBJ, tem-se:

$$E_b(dBJ) = 10\log E_b \tag{2.10}$$

Dado que $T_b = 1/r_b$, onde r_b é a taxa de bit, em bits por segundo, então E_b pode ser reescrito como:

$$E_b = \frac{C}{r_b} \text{ (J/bit)} \tag{2.11}$$

Em dBJ, tem-se:

$$E_b(dBJ) = 10\log\frac{C}{r_b} = 10\log C - 10\log r_b \tag{2.12}$$

A densidade de potência do ruído é a potência do ruído térmico normalizado em uma largura de faixa de 1 Hz (isto é, a potência de ruído presente em uma largura de faixa de 1 Hz). Matematicamente, a densidade de potência do ruído N_0 é dada por:

$$N_0 = \frac{N}{B} \quad \text{(W/Hz)} \tag{2.13}$$

onde N_0 é a densidade de potência do ruído (W/Hz); N é a potência do ruído térmico (W); B é a largura de faixa efetiva do ruído (Hz).

Em dBm, tem-se:

$$N_0(dBm) = 10\log\frac{N}{0.001} - 10\log B = N(dBm) - 10\log B \tag{2.14}$$

Combinando as equações 2.5 e 2.13, temos:

$$N_0 = KT \quad \text{(W)} \tag{2.15}$$

Em dBm, tem-se:

$$N_0(dBm) = 10\log\frac{K}{0.001} + 10\log T \tag{2.16}$$

A relação entre a energia de bit e a densidade de potência de ruído é usada para comparar dois ou mais sistemas de modulação digital que usam diferentes taxas de transmissão (taxa de bit), esquemas de modulação (FSK, PSK, QAM), ou técnicas de codificação (M-níveis). A relação entre a energia de bit pela densidade de potência de ruído é simplesmente a relação da energia de um único bit pela potência de ruído presente em uma largura de faixa de 1 Hz. Então E_b / N_0 normaliza todos os esquemas de modulação multi-fases à uma largura de faixa de ruído comum, permitindo uma comparação mais simples e eficiente na medida de taxa de erro.

Matematicamente, E_b / N_0 é dado por:

$$\frac{E_b}{N_0} = \frac{C / r_b}{N / B} = \frac{C \cdot B}{N \cdot r_b} \tag{2.17}$$

onde E_b / N_0 é a relação entre a energia de bit pela e a densidade de potência de ruído.

Re-arranjando a equação 2.17 tem-se a seguinte expressão:

$$\frac{E_b}{N_0} = \frac{C}{N} \times \frac{B}{r_b} \tag{2.18}$$

onde $\frac{E_b}{N_0}$ é a relação entre a energia de bit e a densidade de potência de ruído; $\frac{C}{N}$ é a relação entre potência da portadora e a potência do ruído térmico; $\frac{B}{r_b}$ é a relação entre largura de faixa e a taxa de bit.

Em dB, tem-se:

$$\frac{E_b}{N_0}(dB) = 10\log\frac{C}{N} + 10\log\frac{B}{r_b} = 10\log E_b - 10\log N_0 \tag{2.19}$$

Da equação 2.18 pode ser visto que a relação E_b / N_0 é simplesmente o produto da relação potência da portadora sobre ruído e a relação largura de faixa sobre a taxa de bit. Também da equação 2.18, pode ser visto que onde a largura de faixa é igual taxa de bit, então $E_b / N_0 = C / N$.

Defini-se [1] que a expressão geral para a probabilidade de erro de bit do sistema PSK com M níveis é dada por:

$$P(e) = \frac{1}{\log_2 M} erf(z) \tag{2.20}$$

onde erf é a função de erro [1]; $z = \text{sen}\frac{\pi}{M}(\sqrt{\log_2 M})(\sqrt{E_b / N_0})$.

A figura 2.6 ilustra a medida de erro para sistemas PSK, 8-PSK e 16-PSK em função de E_b / N_0. Pode-se notar que quanto maior E_b / N_0 (energia de bit sobre densidade de potência de ruído) menor será a $P(e)$.

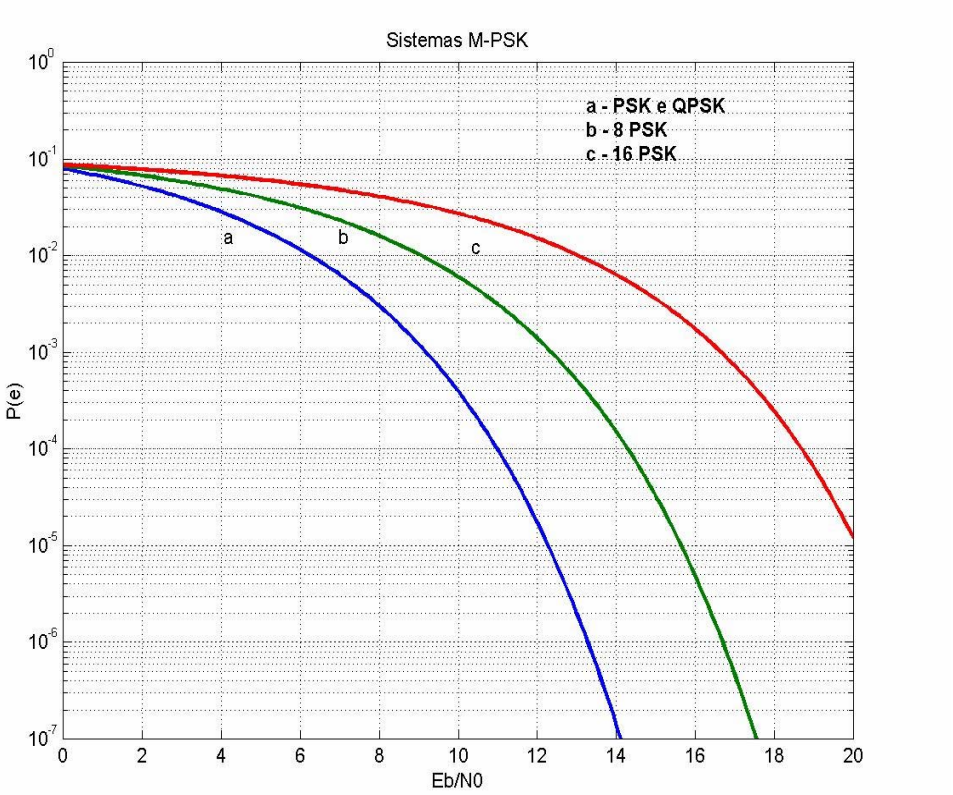


Figura 2.6 – Curvas de taxa de erro para M-níveis PSK

2.5 - Monitoramento de Erros Usando Seqüências de Teste

A figura 2.7 ilustra o diagrama de blocos para obtenção da medida de taxa de erros proposto neste trabalho:

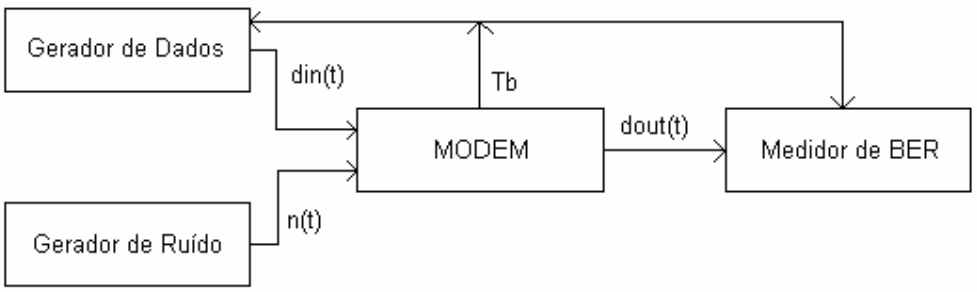


Figura 2.7 – Configuração para medida de taxa de erros

O diagrama de blocos da figura 2.7 mostra o conjunto de dispositivos utilizado para levantamento da Taxa de Erros. Através do Gerador de Dados aplica-se uma seqüência pseudo-aleatória de dados e através do Gerador de Ruídos associado com uma unidade de

filtros programáveis, injeta-se ruído gaussiano. O medidor de BER faz a contagem de bits errados e fornece a Taxa de Erro de Bits.

2.6 – Avaliação de Desempenho de Sistemas de Comunicação Utilizando Códigos Corretores de Erros

O objeto principal deste trabalho é a implementação de um dispositivo para a avaliação de desempenho de sistemas de comunicação utilizando códigos cíclicos redundantes CRC. Neste trabalho foi escolhido o código CRC para implementação do projeto por ser de alta eficiência na detecção de erros na transmissão de dados.

Os códigos cíclicos formam uma sub classe do conjunto de blocos lineares [3]. Os códigos cíclicos têm uma estrutura linear capaz de serem analisados utilizando álgebra linear. Uma vantagem de códigos cíclicos é que entre muitos outros tipos de códigos, estes são facilmente codificados. Os códigos cíclicos apresentam outras duas vantagens [3]: a primeira é que a codificação e o cálculo da síndrome podem ser implementados por registradores de deslocamentos com conexões de realimentação (ou circuitos seqüenciais lineares); e a segunda, é porque eles têm estrutura algébrica, que possibilita encontrar diversos métodos práticos para sua decodificação.

Códigos CRC

Os Códigos Cíclicos Redundantes (CRC) são os mais eficazes na detecção de erros durante a transmissão de dados [1]. Com o uso do código CRC, aproximadamente 99.95% dos erros na transmissão de dados são detectados.

O código CRC mais comum é o CRC-16 [1], que é idêntico ao padrão internacional ITU-T V.41. Com o código CRC-16, 16 bits são usados na seqüência verificadora de bloco (BCS). Essencialmente, o caracter CRC é o resto da divisão de um processo.

A proposta deste trabalho é desenvolver o projeto programável de códigos CRC de até 32 bits. No entanto, convém salientar que há códigos padronizados internacionalmente com um número menor de bits e com eficiência adequada para avaliação do canal de comunicação.

Os códigos mais comuns são: CRC-5, CRC-8, CRC-10, CRC-12, CRC-ITU, CRC-16, CRC-32 com os seguintes polinômios:

CRC-5: $g(D) = D^5 + D^2 + 1$

CRC-8: $g(D) = D^8 + D^2 + D + 1$

CRC-10: $g(D) = D^{10} + D^9 + D^5 + D^4 + D + 1$

CRC-12: $g(D) = D^{12} + D^{11} + D^3 + D^2 + D + 1$

CRC-ITU: $g(D) = D^{16} + D^{12} + D^5 + 1$

CRC-16: $g(D) = D^{16} + D^{15} + D^2 + D + 1$

CRC-32: $g(D) = D^{32} + D^{26} + D^{23} + D^{22} + D^{16} + D^{12} + D^{11} + D^{10} + D^8 + D^7 + D^5 + D^4 + D^2 + D + 1$

No capítulo 3 tem-se uma descrição dos códigos CRC.

CAPÍTULO 3 – MEDIDOR DE TAXA DE ERRO DE BITS – BER, UTILIZANDO CÓDIGO CRC

3.1 – Introdução

A função do código corretor de erro é fornecer mecanismos de detecção e correção de erro(s) em sistemas de comunicações. Há basicamente dois tipos de códigos corretores de erros: os códigos de bloco lineares e os códigos de bloco convolucionais. Neste capítulo é descrito o código de interesse para a implementação de um medidor de taxa de erro de bits; neste caso, tem-se o código CRC (Códigos de Redundância Cíclica).

Os códigos CRC são um subconjunto dos códigos de bloco lineares [3]. A seguir, tem-se uma breve descrição dos mesmos.

3.2 – Código de Bloco Linear

Dado uma mensagem de *k* bits descrita por: $m_0, m_1, \cdots, m_{k-1}$; esta mensagem aplicada em um codificador de bloco linear gera um código de verificação de paridade de $(n-k)$ bits [2]. A soma destas duas seqüências será a nova palavra a ser transmitida pelo sistema de comunicação. A figura 3.1 ilustra esta operação:

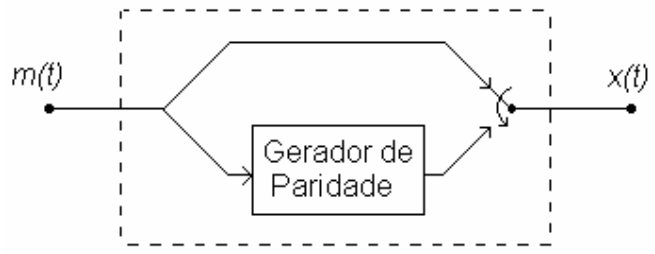


Figura 3.1 – Codificador de Códigos

A saída do decodificador será composta dos bits de mensagem (informação) mais os bits de paridade. A forma como é composta a palavra de saída define códigos sistemáticos e não sistemáticos. Para os códigos sistemáticos há uma separação precisa entre palavras de mensagem e a palavra de paridade. Neste trabalho trata-se com códigos sistemáticos para facilitar o tratamento matemático.

Portanto, define-se [2],

$$x_i = \underbrace{b_0, b_1, \cdots, b_{n-k-1}}_{paridade} \underbrace{m_0, m_1, \cdots, m_{k-1}}_{mensagem} \tag{3.1}$$

o que quer dizer que *x* é função de *b* e *m*.

Os $(n-k)$ bits de paridade, representados por b_i, são somas lineares (aritmética complemento 2) da mensagem de *k* bits, dados por:

$$b_i = p_{i0} \cdot m_0 + p_{i1} \cdot m_1 + \cdots + p_{i,k-1} \cdot m_{k-1} \tag{3.2}$$

onde $i = 0,1,\cdots,n-k-1$ e os coeficientes p_{ij}, dados por:

$$p_{ij} = \begin{cases} 1, & \textit{se } b_i \textit{ depende de } m_j \\ 0, & \textit{caso contrário} \end{cases} \tag{3.3}$$

são coeficientes escolhidos de tal maneira que as $(n-k)$ equações de (3.2) são linearmente independentes; isto é, nenhuma outra equação no conjunto é expressa como a combinação linear de todas as outras.

Utilizando notação matricial, defini-se [2]:

1) vetor mensagem $\underline{m}$ de dimensão $[1 \times k]$:

$$\underline{m} = [m_0, m_1, \cdots, m_{k-1}] \tag{3.4}$$

2) vetor paridade $\underline{b}$ de dimensão $[1 \times (n-k)]$:

$$\underline{b} = [b_0, b_1, \cdots, b_{n-k-1}] \tag{3.5}$$

De acordo com a definição da geração do código de paridade pode-se escrever que:

$$\underline{b} = \underline{m} \cdot \underline{P} \tag{3.6}$$

onde $\underline{P}$ é a matriz de coeficientes definida por:

$$\underline{P} = \begin{bmatrix} p_{00} & p_{10} & \cdots & p_{n-k-1,0} \\ \vdots & \vdots & \ddots & \vdots \\ p_{0,k-1} & p_{1,k-1} & \cdots & p_{n-k-1,k-1} \end{bmatrix} \tag{3.7}$$

que define a estrutura do código.

3) e finalmente o vetor palavra código $\underline{x}$ de dimensão $[1 \times n]$:

$$\underline{x} = [x_0, x_1, \cdots, x_{n-1}] \tag{3.8}$$

que pode ser expresso como:

$$\underline{x} = \left[\underline{b} \ \vdots \ \underline{m}\right] \tag{3.9}$$

Substituindo (3.6) em (3.9) e fatorando temos,

$$\underline{x} = \underline{m} \ \left[\underline{P} \ \vdots \ \underline{I}_k\right] \tag{3.10}$$

onde $\underline{I}_k$ é definida como a matriz identidade de dimensão $k \times k$

$$\underline{I}_k = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{3.11}$$

Uma matriz geradora G de dimensão $k \times n$, é definida em [2] por:

$$\underline{G} = \left[\underline{P} \ \vdots \ \underline{I}_k\right] \tag{3.12}$$

E portando, após o codificador teremos a transmissão do vetor palavra código dado por:

$$\underline{x} = \underline{m} \ \underline{G} \tag{3.13}$$

O vetor transmitido do lado do receptor deverá ser tratado de forma a extrair a mensagem original para gerar uma matriz, neste caso, verificadora de paridade. Esta matriz verificadora de paridade, quando comparada com a matriz obtida no gerador, fornecerá um diagnóstico da palavra recebida, no caso, indicando se há erros e dependendo do código, onde se localiza o erro.

A matriz verificadora de paridade [2] é representada por H, com dimensão $\left[(n-k) \times n\right]$ e é definida como:

$$\underline{H} = \left[\underline{I}_{n-k} \ \vdots \ \underline{P}^T\right] \tag{3.14}$$

onde $\underline{I}_{n-k}$ é a matriz identidade de dimensão $(n-k) \times (n-k)$ e $\underline{P}^T$ é a matriz transposta de $\underline{P}$ com dimensão $(n-k) \times k$.

Aplicando-se a matriz $\underline{H}^T$ ($\underline{H}$ transposta) na expressão transmitida tem-se a expressão que verifica a paridade do sistema [2],

$$\underline{x} \ \underline{H}^T = \underline{m} \ \underline{G} \ \underline{H}^T = 0 \tag{3.15}$$

Desta forma, aplicando-se uma seqüência de dados padrão no transmissor, e, utilizando esta seqüência no receptor, pode-se avaliar o canal de comunicação. A figura 3.2 ilustra a operação para obtenção da medida de erros:

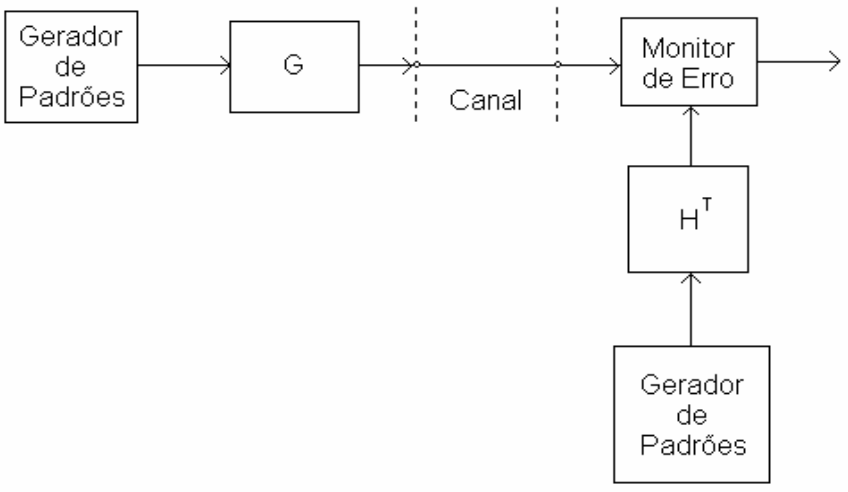


Figura 3.2 – Medida de Erros

A figura 3.2 mostra que a implementação do medidor é dependente da dificuldade de implementação da matriz geradora (G) e da matriz verificadora (H) de paridade. Na classe dos códigos de bloco há um conjunto de códigos mais fáceis de serem implementados. São eles: os códigos cíclicos e os códigos de redundância cíclica.

Códigos Cíclicos

Um código cíclico é definido como um código de bloco linear com a propriedade que: o deslocamento cíclico à direita de qualquer palavra código é uma palavra código.

As palavras códigos geradas podem ser representadas por uma equação polinomial de ordem $(n-1)$ definido em [2] como,

$$x(D) = x_0 + x_1 D + \cdots + x_{n-1} D^{n-1} \tag{3.16}$$

onde *D*, para os sistemas binários podem assumir o valor 0 ou 1, e a potência de *D* representa o deslocamento cíclico de 1 bit à direita. A equação polinomial representa fielmente a palavra código gerada.

Um código cíclico (n,k) é especificado pelo conjunto de $(n-1)$ polinômios da palavra código $x(D)$. Neste conjunto de polinômios (ou palavras códigos) há um polinômio de grau mínimo, representado por $g(D)$, chamado de gerador polinomial. O gerador polinomial $g(D)$ é equivalente a matriz geradora (G) descrita para o código de bloco linear.

Uma mensagem escrita na forma polinomial é descrita por [2],

$$m(D) = m_0 + m_1 D + \cdots + m_{k-1} D^{k-1} \tag{3.17}$$

Na forma sistemática tem-se [2],

$$x(D) = \underbrace{b_0, b_1, \cdots, b_{n-k-1}}_{(n-k)\ bit's\ de\ paridade}, \underbrace{m_0, m_1, \cdots, m_{k-1}}_{k\ bit's\ de\ mensagem}$$

Assim, como definido por Haykin [2], pode-se descrever as palavras códigos de um código cíclico como,

$$x(D) = m(D) \cdot g(D) \tag{3.18}$$

A mensagem transmitida com comprimento de *n* bits é composta da mensagem deslocada de D^{n-k} bits mais os bits de paridade, $b(D)$ definido em [2] por:

$$x(D) = b(D) + D^{n-k} m(D) \tag{3.19}$$

Os bits de paridade são calculados dividindo a mensagem $m(D)$ deslocada de D^{n-k} pelo gerador polinomial como segue:

$$\frac{D^{n-k}m(D)}{g(D)} = a(D) + \frac{b(D)}{g(D)} \tag{3.20}$$

O resto após a divisão é o polinômio de paridade $b(D)$, assim

$$D^{n-k}m(D) = a(D)g(D) + b(D) \tag{3.21}$$

Define-se [7] que todo polinômio gerado é um múltiplo do polinômio gerador, pois $b(D)$ tem grau menor do que $g(D)$, assim o resto de $x(D)/g(D)$, é:

$$\begin{aligned} resto\left[\frac{x(D)}{g(D)}\right] &= \left[\frac{b(D) + D^{n-k}m(D)}{g(D)}\right] = \left[\frac{b(D)}{g(D)} + \frac{g(D)a(D) + b(D)}{g(D)}\right] \\ &= b(D) + b(D) = 0 \end{aligned} \tag{3.22}$$

pois, $b(D) = -b(D)$

O polinômio recebido $r(D)$ contém o polinômio transmitido mais um polinômio de erro que representa o erro ocorrido durante a transmissão dos dados através canal, então

$$r(D) = x(D) + e(D) \tag{3.23}$$

O receptor divide o polinômio recebido por $g(D)$ e verifica o resto:

$$\begin{aligned} resto\left[\frac{r(D)}{g(D)}\right] &= resto\left[\frac{x(D) + e(D)}{g(D)}\right] \\ &= resto\left[\frac{x(D)}{g(D)}\right] + resto\left[\frac{e(D)}{g(D)}\right] \end{aligned} \tag{3.24}$$

De (3.22) tem-se,

$$resto\left[\frac{r(D)}{g(D)}\right] = resto\left[\frac{e(D)}{g(D)}\right] \tag{3.25}$$

Um erro é detectado se $e(D)$ não é um múltiplo de $g(D)$ e assim tem-se,

$$resto\frac{e(D)}{g(D)} \neq 0 \tag{3.26}$$

Assim, o circuito para calcular o resto, e conseqüentemente a verificação do erro tem a mesma estrutura do gerador polinomial.

Medida de Erros com Código Cíclico

O medidor de erros com código cíclico é composto de três blocos: o gerador de dados aleatório no transmissor, o decodificador do sinal recebido e o gerador de dados local no receptor [8].

3.3 – Gerador de Dados Aleatório

O gerador de dados aleatório é um gerador polinomial de código cíclico realimentado, que tem como função gerar todos os padrões de n bits (0's e 1's) de uma seqüência de comprimento n.

A forma genérica deste gerador é

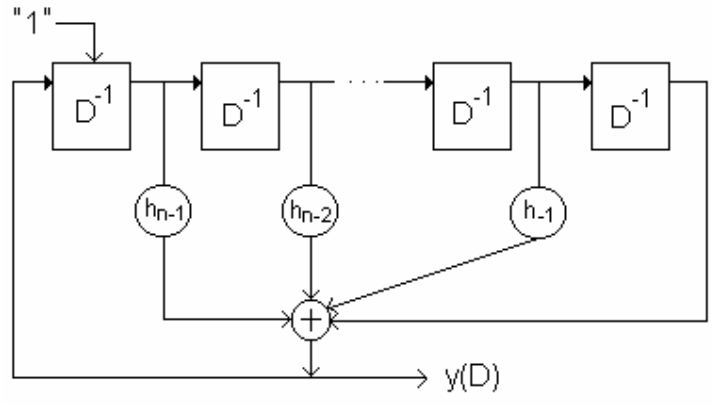


Figura 3.3 – Gerador de seqüência $y(D)$

Pode-se observar que há uma realimentação para a posição D^{n-1} e uma entrada para assegurar uma condição aleatória na malha, pois caso contrário, como a saída do somador é zero, para todas as entradas iguais a zero teríamos uma condição de estabilidade.

A saída do gerador é:

$$y(D) = D^{n-1}h_{n-1} + D^{n-2}h_{n-2} + \cdots + D^{-1} \tag{3.27}$$

Este dispositivo gera $n-1$ padrões de sinais, ou seja, todas as seqüências de 1's e 0's a menos do padrão todo zero.

3.4 – Decodificador do Sinal Recebido

O decodificador aplicará a operação inversa no sinal recebido.

A forma geral do decodificador é dada por,

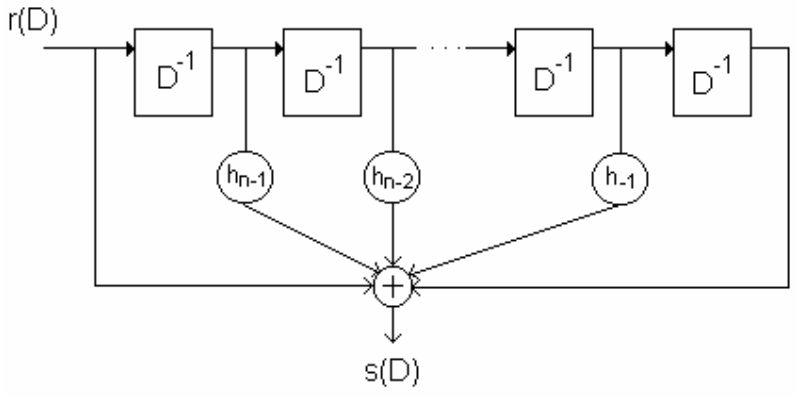


Figura 3.4 – Decodificador do sinal recebido $r(D)$

O sinal recebido é dado por,

$$r(D) = y(D) + e(D) \tag{3.28}$$

O decodificador realiza a operação inversa no sinal transmitido.

$$\begin{aligned} s(D) &= y^{-1}[y(D) + e(D)] = y^{-1}[y(D)] + y^{-1}[e(D)] \\ &= y^{-1}[e(D)] \end{aligned} \tag{3.29}$$

A seguir, este sinal é aplicado no gerador polinomial cíclico local para a verificação de paridade como ilustrado na figura 3.5:

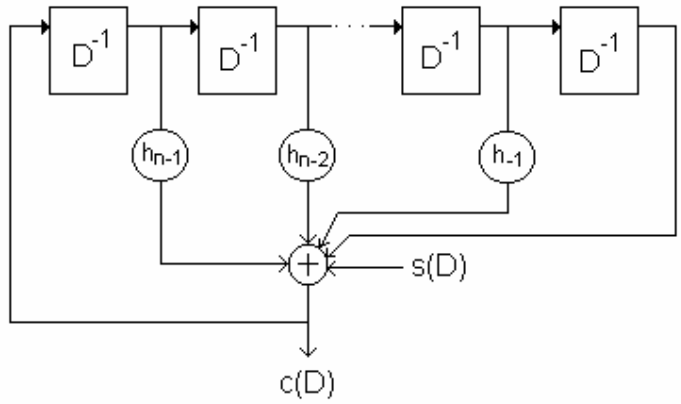


Figura 3.5 – Gerador polinomial cíclico local

Assim, a saída de erro será,

$$\begin{aligned} c(D) &= y^{-1}[s(D)] = y^{-1}[y^{-1}(e(D))] \\ &= e(D) \end{aligned} \tag{3.30}$$

Onde $c(D) \neq 0$ representa uma indicação de erro ocorrido na seqüência pseudo-aleatória transmitida.

CAPÍTULO 4 – IMPLEMENTAÇÃO E TESTES DE UM MEDIDOR DE TAXA DE ERRO DE BITS UTILIZANDO LINGUAGEM VHDL

4.1 - Introdução

Este capítulo apresenta o projeto de um dispositivo lógico para a medida de taxa de erro de bits. O projeto deste dispositivo pode utilizar uma das diversas técnicas disponíveis atualmente: circuito lógico discreto em famílias de dispositivos TTL (Transistor Transistor Logic) ou até mesmo ECL (Emitter Coupled Logic) para altas velocidades; software em um sistema microprocessado e dispositivos lógicos programáveis, utilizando CPLD (Complex Programmable Logic Devices) ou FPGA (Field Programmable Gate Array). Optou-se pela implementação em CPLD [9] devido às facilidades de ferramentas de desenvolvimento, tal como, projeto do circuito em linguagem de programação de hardware de alto nível VHDL (Very High Speed Integrated Circuit Hardware Description Language) e simulador embutido na ferramenta. Algumas vantagens deste projeto são: baixo custo para implementação, alta confiabilidade na detecção de erros e necessidade de baixa potência para funcionamento do circuito.

4.2 – Sistema de Medida de BER

O diagrama de blocos da figura 4.1 ilustra o esquema básico do sistema de avaliação de funcionamento do *Medidor de Taxa de Erro de Bit* (BER meter).

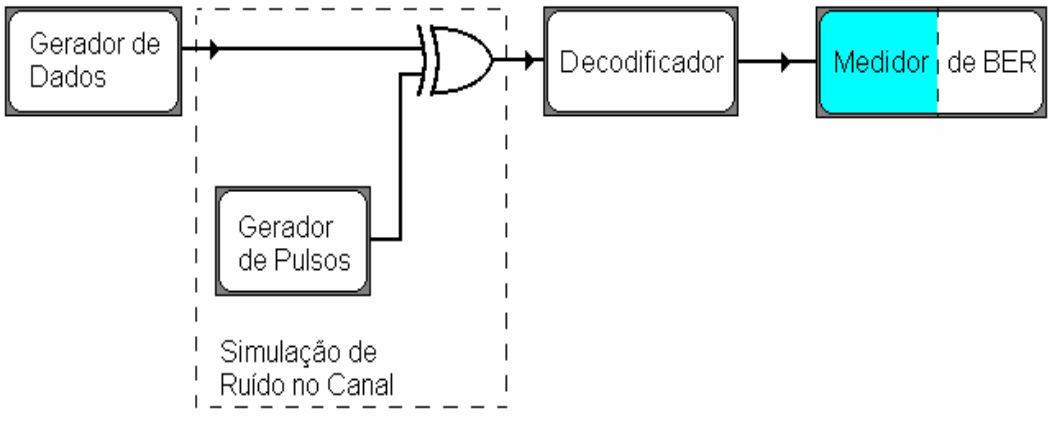


Figura 4.1- Diagrama de blocos lógicos básicos de um sistema de comunicação que utiliza em sua estrutura um Medidor de Taxa de Erro de Bit (BER meter)

A seguir são descritos os blocos da figura 4.1.

O *Gerador de Dados* produz uma seqüência de dados pseudo-aleatória. O polinômio gerador para esta seqüência é selecionado dentro de um conjunto previamente implementado. A implementação é baseada no conjunto padronizado pelo ITU-T da tabela extraída de [10].

O *Canal* é o meio físico por onde a seqüência produzida pelo gerador de dados percorrerá até chegar ao destino.

Para avaliar o dispositivo proposto, utiliza-se um modelo de canal simétrico binário – BSC (Binary Symetric Channel) tal como ilustrado na figura 4.2:

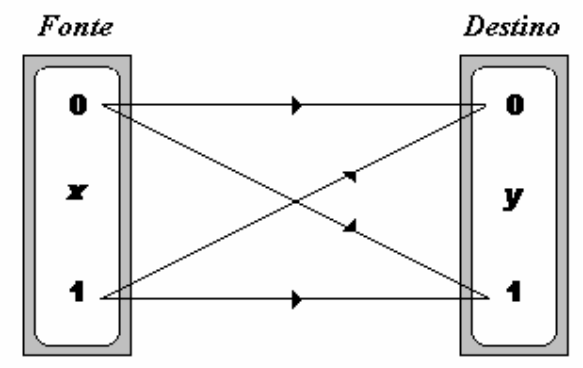


Figura 4.2 – Representação de Canal Digital Binário

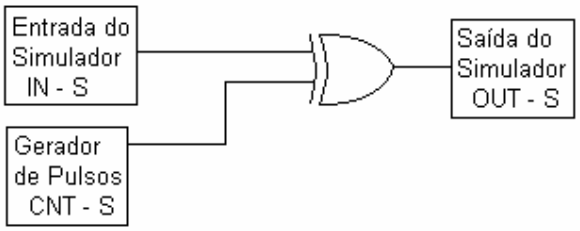
Um erro neste canal é caracterizado pela inversão do sinal digital ($1 \rightarrow 0$ e $0 \rightarrow 1$) entre a entrada e a saída [2].

Na teoria, pode-se avaliar este canal pela equação da Probabilidade de Erro $P(e)$. A equação de erro do canal é dada por:

$$P(erro) = P[y = 0 / x = 1] \cdot P[x = 1] + P[y = 1 / x = 0] \cdot P[x = 0] \qquad (4.1)$$

Na prática, este canal é avaliado pela medida de taxa de erro de bits – BER (Bit Error Rate). Ou seja, um erro no canal ocorre quando há inversão de bit ocasionado por algum processo físico durante a transmissão de dados. A duração desta inversão implica na ocorrência de um ou mais bits errados. Para simular esta inversão é utilizada a porta lógica ou-exclusivo, tal como ilustrado na figura 4.1.

A porta ou-exclusivo tem como característica a inversão do dado controlada por uma das entradas aplicando nível lógico 0 ou 1.



IN - S	CNT - S	OUT - S
0	0	0
1	0	1
0	1	1
1	1	0

Figura 4.3 – Simulador de inversão de bit

Ou seja, quando CNT-S é igual a zero, implica que o dado passa pelo canal sem alteração e quando CNT-S é igual a um, implica na inversão do dado quando o mesmo passa pelo canal, ou seja, temos a simulação de erro.

A entrada de erro é simulada por um gerador pseudo-aleatório com comprimento longo (figura 4.4). A cada *x* segundos ocorrerá a inversão de onda durante *y* segundos onde $y \geq Tb$, sendo *Tb* o tempo de duração de um bit, para que se tenha um ou dois erros a cada *x* segundos. Então a distribuição de erro durante este tempo será uniforme.

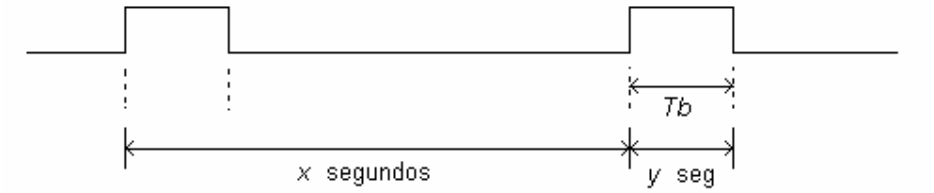


Figura 4.4 – Forma de onda da entrada de erro

O *Decodificador* recebe os dados enviados através do canal e calcula a seqüência invertida gerada no gerador de dados com o(s) erro(s) inserido(s) (se houver), obtendo assim uma função da seqüência transmitida com erro(s) ou sem erro(s). Se não houver erros, a saída do decodificador será zero. Se houver erros, a forma de onda de saída do decodificador será uma função do(s) erro(s) inserido(s) tal como descrito no item 4.3.

O *Medidor de BER* registra os bits de erros que foram inseridos no canal tal como descrito no item 4.4. Pode-se definir BER pela seguinte fórmula:

$$BER = \frac{número\ de\ bits\ errados\ recebidos\ em\ um\ período\ de\ tempo\ definido}{número\ total\ de\ bits\ transmitidos\ sobre\ o\ período\ de\ tempo\ definido}$$

4.3 - Bloco1 – Gerador de Dados

O primeiro bloco é o gerador de dados. Este gerador produz uma seqüência de dados pseudo-aleatória, definido em [8], de comprimento:

$$g(D) = 2^m - 1 \tag{4.2}$$

onde *m* é o maior grau do polinômio gerador ou o número de flip-flops utilizados no circuito gerador.

O dispositivo eletrônico gerador de seqüência de dados pseudo-aleatória é implementado com registradores de deslocamento realimentados onde as conexões implementam um polinômio de grau *m*, que geram uma seqüência digital com a propriedade: quantidades iguais, ou quase iguais de "0"s e "1"s [8]. Os pontos de realimentação são os indicados pelo polinômio gerador. O gerador é constituído por flip-flops tipo D interligados entre si. A saída Q de cada flip-flop é ligada à entrada D do próximo. São usadas portas lógicas do tipo ou-exclusivo. O circuito da figura 4.5 tem como função gerar uma seqüência de dados pseudo-aleatória que é entregue na saída do mesmo, baseado em um polinômio gerador $g(D)$. O relógio da seqüência de bits é dado pela entrada de *clock*.

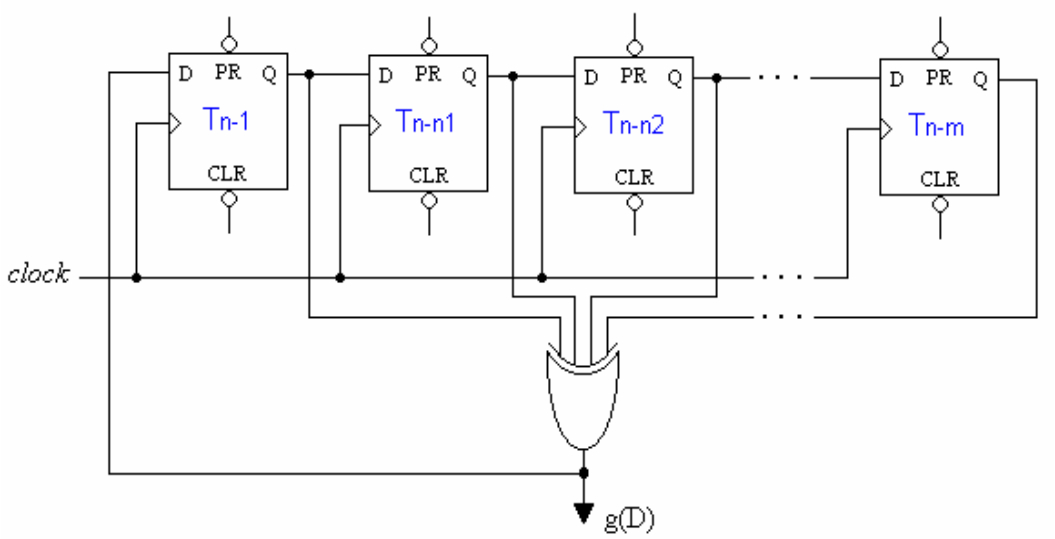


Figura 4.5 – Gerador de Seqüência de dados

A figura 4.6 ilustra a implementação de um gerador pseudo-aleatório de dados, que utiliza o polinômio gerador de grau 4:

$$g(D) = D^4 + D + 1 \tag{4.3}$$

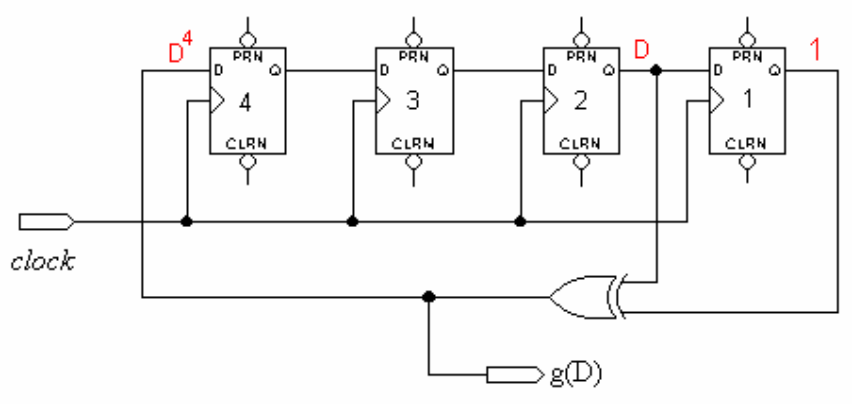


Figura 4.6 – Gerador de Seqüência baseado no polinômio $g(D) = D^4 + D + 1$

O circuito para implementação do gerador da figura 4.6 utiliza 4 flip-flops tipo D e lógica combinacional. A saída do primeiro flip-flop é ligada à entrada do segundo flip-flop. A saída do segundo flip-flop é ligada à entrada do terceiro. A saída do terceiro é ligada à entrada do quarto flip-flop. A saída do terceiro e quarto flip-flops, também são as entradas da porta ou-exclusivo. A saída da porta ou-exclusivo é ligada à entrada do primeiro flip-flop. As entradas da porta ou-exclusivo estão ligadas às saídas do terceiro e quarto flip-flops e a saída da porta ou-exclusivo está ligada à entrada do primeiro flip-flop por quê são os passos de realimentação indicados pelo polinômio da equação 4.3 $g(D) = D^4 + D + 1$.

A forma de onda resultante em $g(D)$ pode ser vista na figura 4.7:

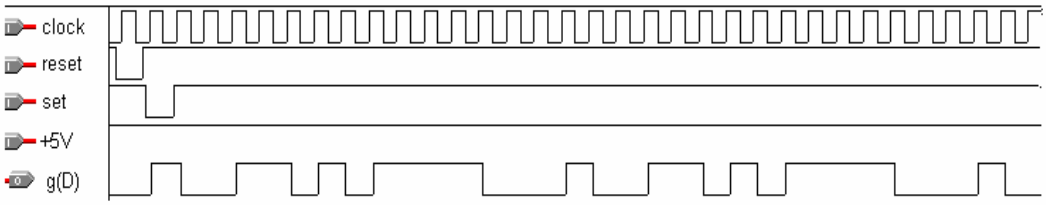


Figura 4.7 – Forma de onda da seqüência $g(D)$

A primeira forma de onda é o gerador de *clock*. A segunda forma de onda é a entrada *reset* que deve ser ativada (nível lógico baixo) somente no primeiro pulso de clock, entregando nível lógico baixo em todos os flip-flops e permanecendo desativada (nível lógico alto) o tempo restante. A terceira forma de onda é a entrada *set* que deve ser ativada (nível lógico baixo) somente no segundo pulso de clock, entregando nível lógico alto no último flip-flop do bloco gerador de dados. Nos demais pulsos de clock a entrada *set* permanece desativada (nível lógico alto). A quarta entrada é a *alimentação* fixada em *+5 Volts* (nível lógico alto) para permanecer desativada. E a quinta forma de onda, é a saída resultante

em $g(D)$. Da equação 4.2, para o polinômio gerador $g(D) = D^4 + D + 1$ (4.3), o comprimento da seqüência $g(D)$ é de $2^4 - 1 = 15$ bits. Note que após uma seqüência de 15 bits, a mesma seqüência começa a se repetir. Tal seqüência tem quantidades quase iguais de "0"s e "1"s [8]. Desta forma, consegue-se entregar na saída do circuito gerador uma seqüência de dados pseudo-aleatória utilizada na avaliação de desempenho do sistema de comunicação.

Outro polinômio gerador de seqüência de dados pseudo-aleatória é obtido através da implementação do circuito da figura 4.8, utilizando o polinômio de grau 3:

$$g(D) = D^3 + D + 1 \tag{4.4}$$

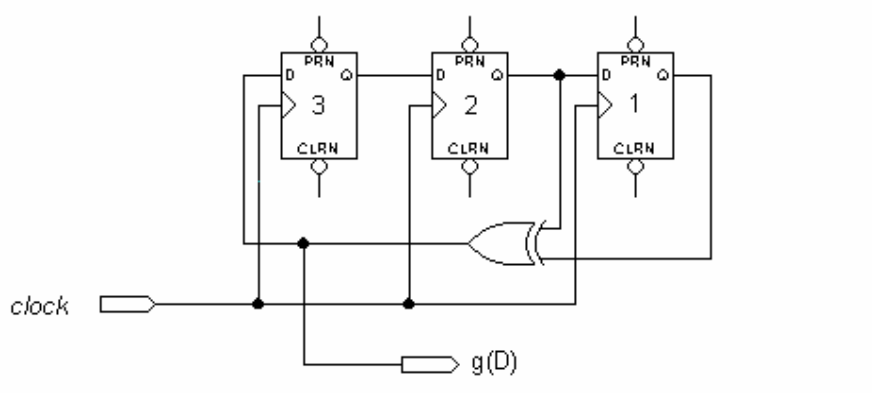


Figura 4.8 – Gerador de seqüência baseado no polinômio $g(D) = D^3 + D + 1$

O circuito para implementação do gerador da figura 4.8 utiliza 3 flip-flops tipo D e lógica digital. A saída do primeiro flip-flop é ligada à entrada do segundo flip-flop. A saída do segundo flip-flop é ligada à entrada do terceiro. A saída do segundo e terceiro flip-flops, também são as entradas da porta ou-exclusivo. A saída da porta ou-exclusivo é ligada à entrada do primeiro flip-flop. As entradas da porta ou-exclusivo estão ligadas às saídas do segundo e terceiro flip-flops e a saída da porta ou-exclusivo está ligada à entrada do primeiro flip-flop por quê são os passos de realimentação indicados pelo polinômio da equação 4.4 $g(D) = D^3 + D + 1$.

A forma de onda resultante em $g(D)$ pode ser vista na figura 4.9:

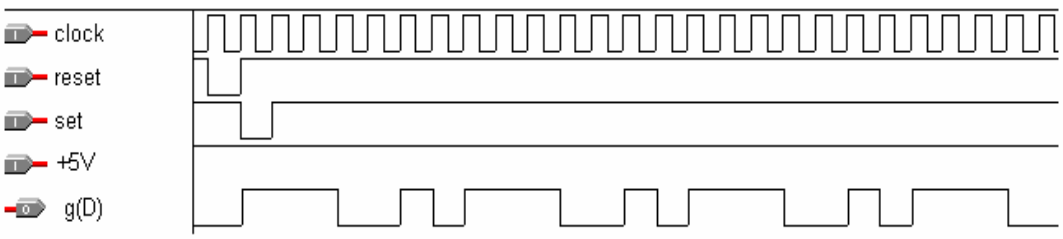


Figura 4.9 – Forma de onda da seqüência $g(D)$

A primeira forma de onda é o gerador de *clock*. A segunda forma de onda é a entrada *reset* que deve ser ativada (nível lógico baixo) somente no primeiro pulso de clock, entregando nível lógico baixo na saída de todos os flip-flops e permanecendo desativada (nível lógico alto) o tempo restante. A terceira forma de onda é a entrada *set* que deve ser ativada (nível lógico baixo) somente no segundo pulso de clock, entregando nível lógico alto na saída do último flip-flop do bloco gerador de dados. Nos demais pulsos de clock a entrada *set* permanece desativada (nível lógico alto). A quarta entrada é a *alimentação* fixada em *+5 Volts* (nível lógico alto) para permanecer desativada. E a quinta forma de onda, é a saída resultante em $g(D)$. Da equação 4.2, para o polinômio gerador $g(D) = D^3 + D + 1$ (4.4), o comprimento da seqüência $g(D)$ é de $2^3 - 1 = 7$ bits. Note que após uma seqüência de 7 bits, a mesma seqüência começa a se repetir. Desta forma, consegue-se entregar na saída do circuito gerador uma seqüência de dados pseudo-aleatória com 7 bits de comprimento utilizada na avaliação de desempenho do sistema de comunicação.

4.4 - Bloco 2 – Decodificador

O decodificador é um registrador de deslocamento que recebe a seqüência de dados produzida pelo gerador pseudo-aleatório com inserção de erro no canal. O decodificador é similar ao bloco gerador, constituído pelo mesmo número de flip-flops tipo D interligados entre si como no gerador. O registrador de deslocamento do decodificador irá conter exatamente a mesma seqüência que no registrador de deslocamento do gerador para uma transmissão sem erros [8]. A saída Q de cada flip-flop é ligada à entrada D do próximo. São usadas portas lógicas do tipo ou-exclusivo. A diferença entre o gerador de seqüência pseudo-aleatória e o decodificador está na entrada do primeiro flip-flop do gerador e na entrada do primeiro flip-flop do decodificador. A entrada do primeiro flip-flop do gerador está ligada à saída da porta ou-exclusivo do próprio gerador, e a entrada do primeiro flip-flop do decodificador está ligada à saída do canal. O decodificador executa o inverso da seqüência produzida pelo gerador com os erros inseridos no canal. O relógio da seqüência de bits é dado pela entrada de clock. Como a seqüência recebida pelo decodificador é $g(D) \oplus e$, a saída do decodificador é dada por:

$$g^{-1}\big(g(D) + e\big) = g^{-1}(g(D)) + g^{-1}(e) = 0 + g^{-1}(e) = g^{-1}(e) \qquad (4.5)$$

Desta forma, a saída do gerador então será uma função inversa do erro inserido. Se não ocorrer nenhum erro durante a transmissão dos dados, a saída do decodificador será zero.

O esquema de um decodificador genérico é ilustrado na figura 4.10:

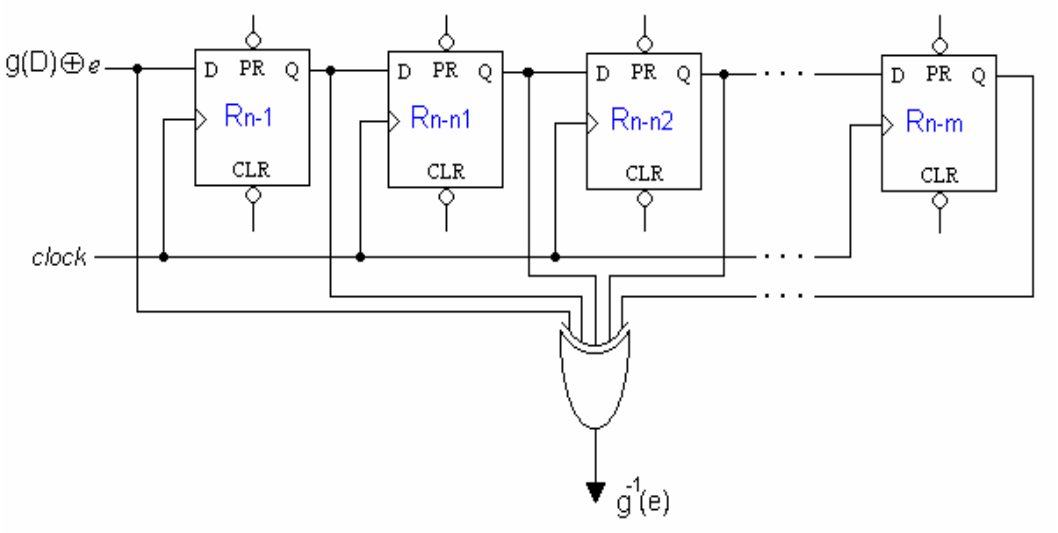


Figura 4.10 – Decodificador de seqüência de dados

Para o decodificador, seja a seqüência $g(D)$ da figura 4.7, gerada utilizando o polinômio da equação 4.3 $g(D) = D^4 + D + 1$. Após a passagem da seqüência $g(D)$ através do canal, na saída deste tem-se $g(D) \oplus e$. Se nenhum erro é inserido, a saída do canal será a seqüência ilustrada na figura 4.11:

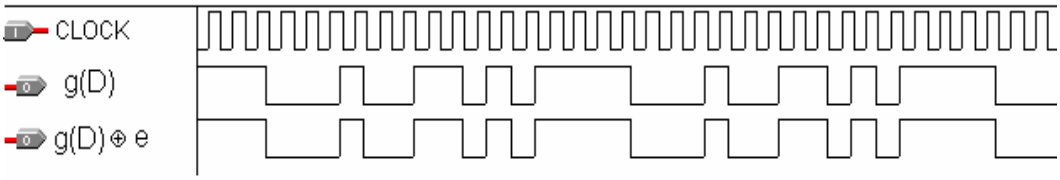


Figura 4.11 – Forma de onda da seqüência $g(D) \oplus e$

Esta seqüência entra no decodificador da figura 4.12 e na saída tem-se $g^{-1}(e)$ conforme ilustrado na forma de onda da figura 4.13:

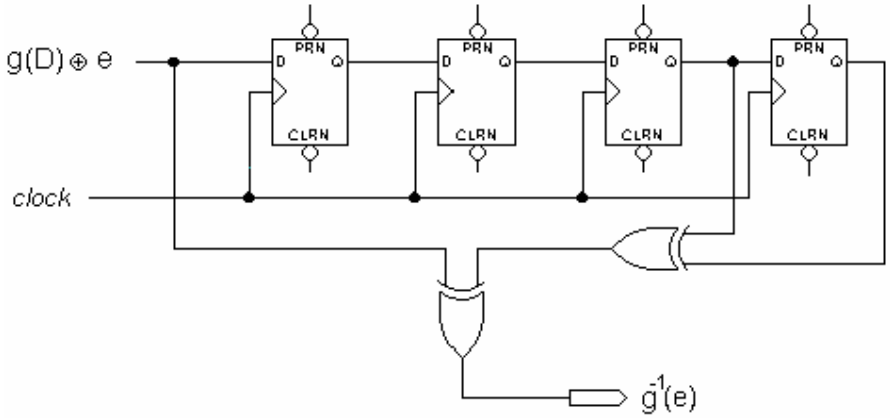


Figura 4.12 – Decodificador baseado no polinômio $g(D) = D^4 + D + 1$

O circuito para implementação do decodificador da figura 4.12 utiliza 4 flip-flops tipo D e lógica digital. A saída do primeiro flip-flop é ligada à entrada do segundo flip-flop. A saída do segundo flip-flop é ligada à entrada do terceiro. A saída do terceiro é ligada à entrada do quarto flip-flop. A saída do terceiro e quarto flip-flops, também são as entradas da primeira porta ou-exclusivo. A saída da primeira porta ou-exclusivo é a entrada da segunda porta ou-exclusivo que tem a outra entrada ligada à saída do canal. Na saída da segunda porta ou-exclusivo tem-se a seqüência $g^{-1}(e)$ como ilustrado na figura 4.13:

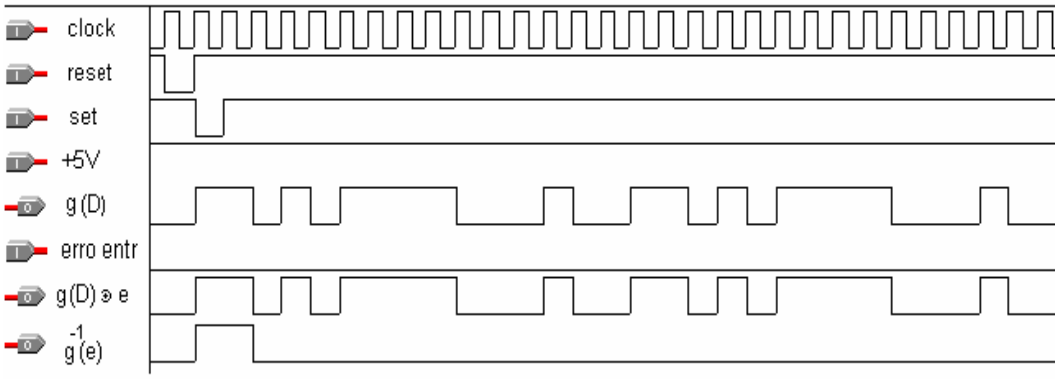


Figura 4.13 – Forma de onda da saída do decodificador que utiliza o polinômio gerador $g(D) = D^4 + D + 1$ *sem nenhuma inserção de erro*

A primeira forma de onda é o gerador de *clock*. A segunda forma de onda é a entrada *reset* que deve ser ativada (nível lógico baixo) somente no primeiro pulso de clock, permanecendo desativada (nível lógico alto) o tempo restante. A terceira forma de onda é a entrada *set* que deve ser ativada (nível lógico baixo) somente no segundo pulso de clock, permanecendo desativada (nível lógico alto) no tempo restante. A quarta entrada é a *alimentação* fixada em *+5 Volts* (nível lógico alto) para permanecer desativada. A quinta forma de onda, é a saída resultante em $g(D)$, ou seja, é a seqüência de dados pseudo-aleatória que está sendo transmitida para avaliar o desempenho do sistema de comunicação. A sexta forma de onda é a *entrada de erro* que está desativada. A sétima forma de onda é a *saída do canal* $g(D) \oplus e$, que neste caso é idêntica à forma de onda do gerador, pois nenhum erro foi inserido. E a oitava forma de onda é a *saída do decodificador* $g^{-1}(e)$, que neste caso é zero pois nenhum erro foi inserido.

4.5 - Bloco 3 – Medidor de BER

4.5.1 - Características do Medidor de BER

1) Gerador de seqüência pseudo-aleatória, decodificador e medidor de taxa de erro de bit para avaliar o desempenho dos sistemas de comunicação digital.
2) Polinômio de comprimento e realimentação programável para gerar qualquer seqüência pseudo-aleatória até $2^{32}-1$ bits; 32 passos de realimentação podem ser escolhidos.
3) Comprimento de seqüência programável, definido pelo usuário para geração de qualquer seqüência repetitiva de até 32 bits de comprimento.

Tabela - Seqüências de testes digitais usados na norma ITU-T definido em [10]:

Comprimento da seqüência (bits)	Zeros Consecutivos	Normas	Uso da seqüência
$2^{09}-1$	8	O.153	Medida de erro em circuitos de dados de taxa de bit até 14 400 bit/s
$2^{11}-1$	10	O.152	Medida de erro em circuitos de dados de taxa de bit até 64 kbit/s and N × 64 kbit/s
$2^{15}-1$	15	O.151	Medida de erro em circuitos de dados de taxa de bit de 1544, 2048, 6312, 8448, 32 064 and 44 736 kbit/s
$2^{20}-1$	19	O.153	Medida de erro em circuitos de dados de taxa de bit até 72 kbit/s
$2^{20}-1$	14	O.151	Medida de erro em circuitos de dados de taxa de bit de 1544, 6312, 32 064 and 44 736 kbit/s
$2^{23}-1$	23	O.151	Medida de erro em circuitos de dados de taxa de bit de 34 368 and 139 264 kbit/s
$2^{29}-1$	29	–	Medida de tarefa específica (ex:medida de atraso à altas taxas de bits)
$2^{31}-1$	31	–	Medida de tarefa específica (ex:medida de atraso à altas taxas de bits)

Tabela 4.1 – Seqüências de testes digitais

4.5.2 – Estrutura do Medidor de BER

O bloco 4 é o registrador de erro. Conforme descrito no capítulo 3, o medidor de BER pode ser implementado como ilustrado na figura 4.14. O dispositivo eletrônico medidor de

taxa de erro de bit é implementado com registradores de deslocamento. Os pontos de realimentação são indicados pelo polinômio gerador. O registrador de erro é constituído por flip-flops tipo D interligados entre si. A saída Q de cada flip-flop é ligada à entrada D do próximo. São usadas portas lógicas do tipo ou-exclusivo. O relógio da seqüência de bits é dado pela entrada de clock.

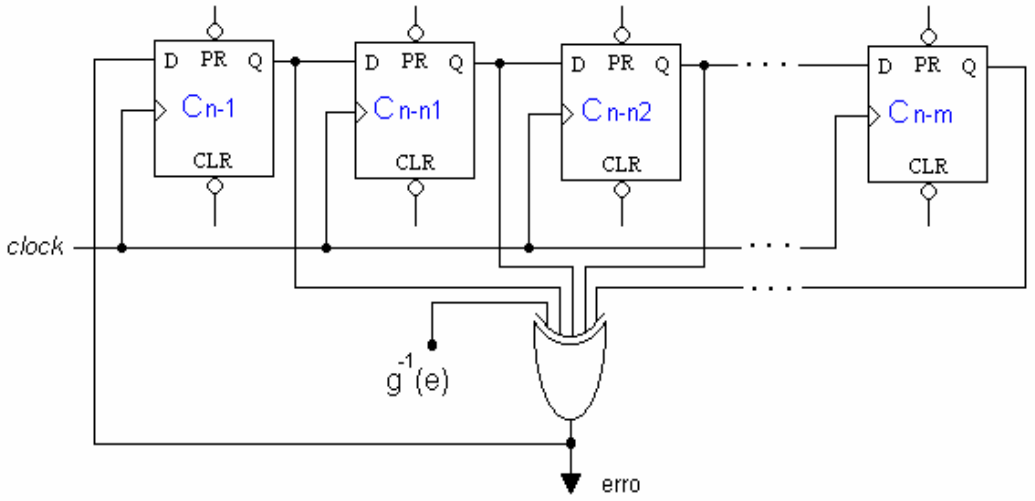


Figura 4.14 – Registrador de erro de bit

A seqüência de dados entregue pelo decodificador, quando comparada com a seqüência de dados gerada pelo registrador de erro através da porta ou-exclusivo, entrega um diagnóstico da seqüência de dados recebida. Este circuito permite que o erro ocorrido durante a transmissão de uma seqüência de dados possa ser detectado em sua saída, obtendo a medida de taxa de erro de bits nos sistemas de comunicação.

Para o polinômio gerador da equação 4.3 $g(D) = D^4 + D + 1$, o circuito registrador de erro de bit pode ser ilustrado pela figura 4.15:

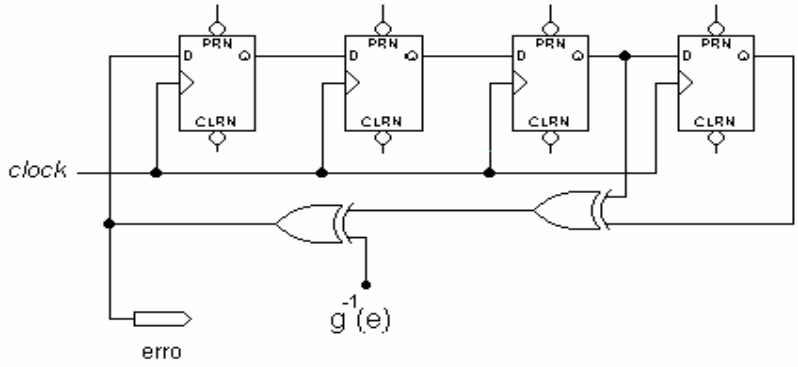


Figura 4.15 – Registrador de erro de bit utilizando polinômio $g(D) = D^4 + D + 1$

O circuito para implementação do medidor de BER da figura 4.15 utiliza 4 flip-flops tipo D e lógica digital. A saída do primeiro flip-flop é ligada à entrada do segundo flip-flop. A saída do segundo flip-flop é ligada à entrada do terceiro. A saída do terceiro é ligada à entrada do quarto flip-flop. A saída do terceiro e quarto flip-flops, também são as entradas da primeira porta ou-exclusivo. A saída da primeira porta ou-exclusivo é a entrada da segunda porta ou-exclusivo que tem a outra entrada ligada à saída do decodificador. Na saída da segunda porta ou-exclusivo tem-se a seqüência de erro detectada. A saída da segunda porta ou-exclusivo está ligada à entrada do primeiro flip-flop por quê são os passos de realimentação indicados pelo polinômio da equação 4.3 $g(D) = D^4 + D + 1$.

Se nenhum erro for inserido no canal, a saída do registrador de erro será zero, conforme ilustrado na figura 4.16:

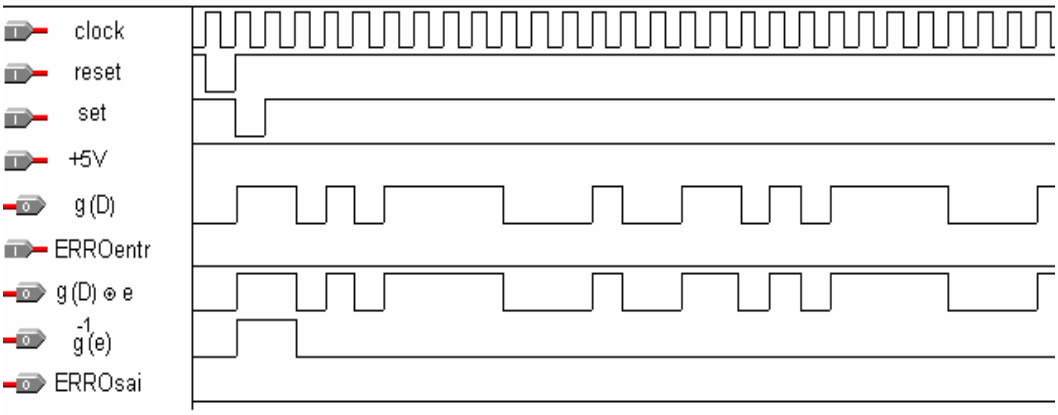


Figura 4.16 – Forma de onda de saída do registrador de erro que utiliza polinômio gerador $g(D) = D^4 + D + 1$ *sem nenhuma inserção de erro*

A primeira forma de onda é o gerador de *clock*. A segunda forma de onda é a entrada *reset* que deve ser ativada (nível lógico baixo) somente no primeiro pulso de clock, permanecendo desativada (nível lógico alto) o tempo restante. A terceira forma de onda é a entrada *set* que deve ser ativada (nível lógico baixo) somente no segundo pulso de clock, permanecendo desativada (nível lógico alto) no tempo restante. A quarta entrada é a *alimentação* fixada em *+5 Volts* (nível lógico alto) para permanecer desativada. A quinta forma de onda, é a saída resultante em $g(D)$, ou seja, a seqüência de dados pseudo-aleatória gerada pelo polinômio $g(D) = D^4 + D + 1$. A sexta forma de onda é a *entrada de erro* que está desativada. A sétima forma de onda é a *saída do canal* $g(D) \oplus e$, que neste caso é idêntica à forma de onda do gerador, pois nenhum erro foi inserido. A oitava forma de onda é a *saída do*

decodificador $g^{-1}(e)$, que neste caso é zero, pois nenhum erro foi inserido. E a nona forma de onda é *a saída do registrador de erro* que também é zero, pois nenhum erro foi inserido.

4.6 – Exemplo Completo de um Sistema Medidor de BER

A figura 4.17 ilustra o esquema completo (*gerador, inserção de erro no canal, decodificador* e *registrador de erro*) de um sistema de comunicação de dados utilizando o polinômio gerador $g(D) = D^4 + D + 1$, simulado pelo software MAX+PLUSII, (ALTERA) [11]. Este circuito combina os três blocos apresentados anteriormente: gerador de dados, decodificador e medidor de BER (gerador de dados local no receptor). O circuito da figura 4.17 é um exemplo específico de um medidor de taxa de erro de bits, onde se pode provar a eficiência na detecção do erro simulado em um sistema de comunicação.

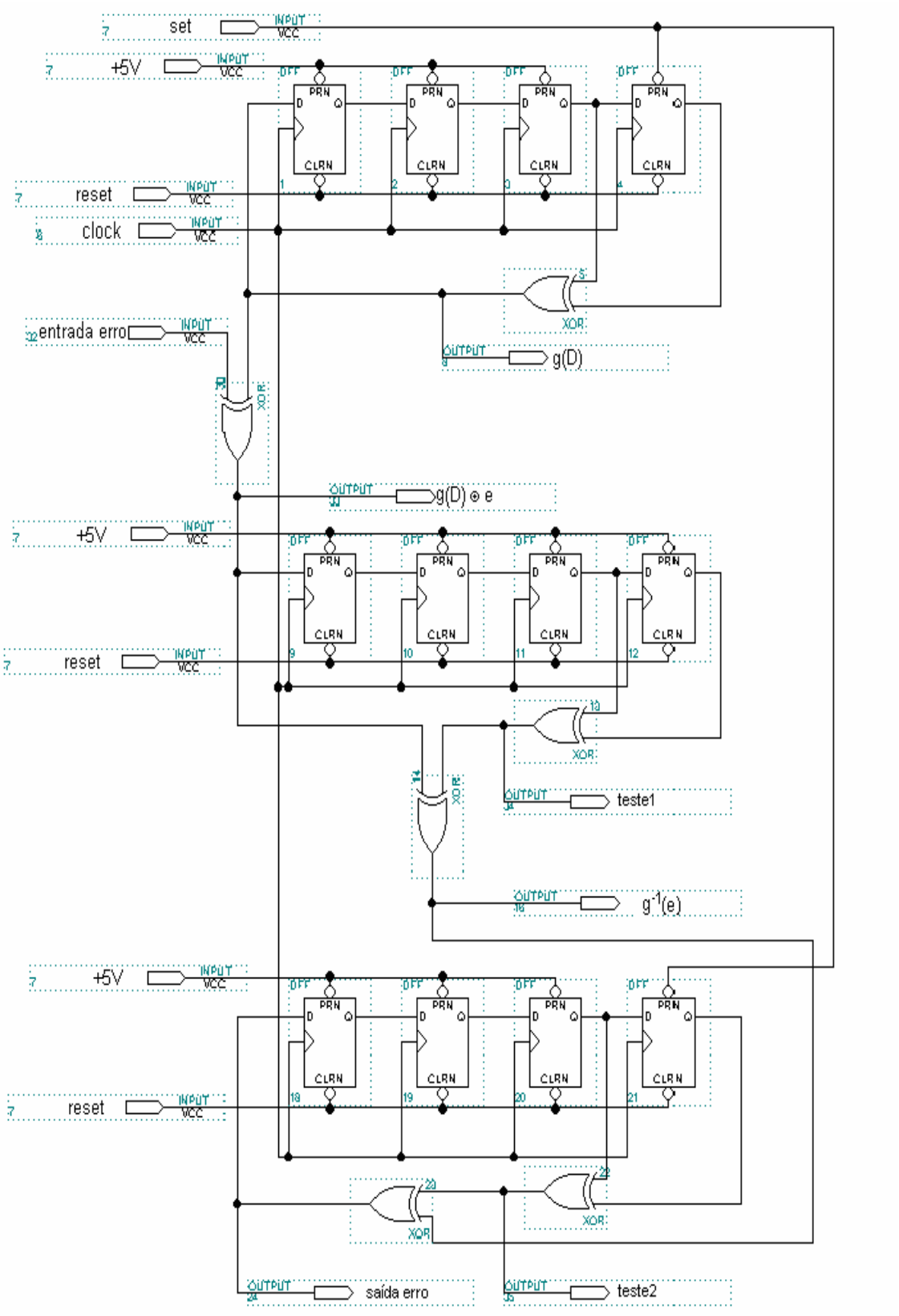


Figura 4.17 – Esquema completo (gerador, decodificador e registrador de erro) de um sistema de comunicação de dados que utiliza polinômio $g(D) = D^4 + D + 1$

A figura 4.18 ilustra as formas de onda do circuito da figura 4.17:

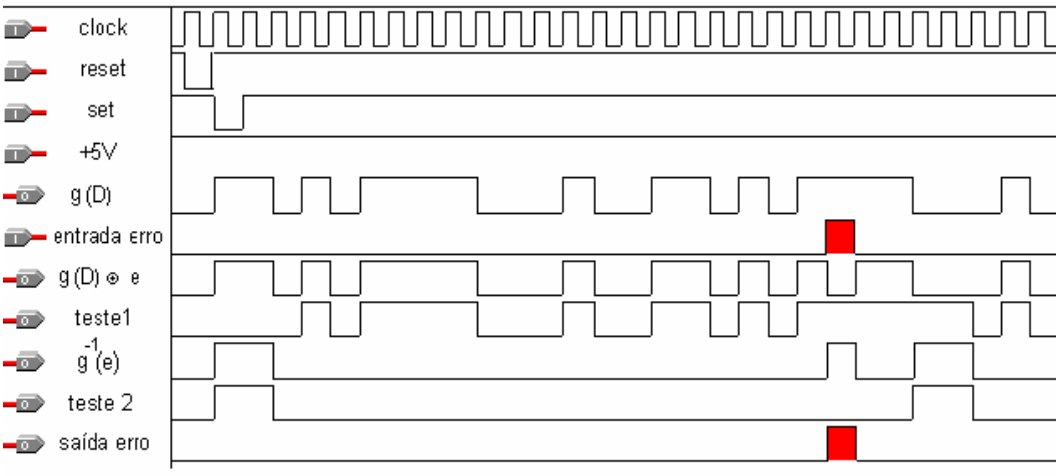


Figura 4.18 – Forma de onda do esquema completo de comunicação de dados utilizando polinômio $g(D) = D^4 + D + 1$

A primeira forma de onda é o gerador de *clock*. A segunda forma de onda é a entrada *reset* que deve ser ativada (nível lógico baixo) somente no primeiro pulso de clock, entregando nível lógico baixo na saída de todos os flip-flops e permanecendo desativada (nível lógico alto) o tempo restante. A terceira forma de onda é a entrada *set* que deve ser ativada (nível lógico baixo) somente no segundo pulso de clock, entregando nível lógico alto na saída do último flip-flop do bloco gerador de dados e do bloco registrador de erro. Nos demais pulsos de clock a entrada *set* permanece desativada (nível lógico alto). A quarta entrada é a *alimentação* fixada em *+5 Volts* (nível lógico alto) para permanecer desativada. A quinta forma de onda, é a saída resultante em $g(D)$, ou seja, a seqüência de dados pseudo-aleatória gerada pelo polinômio $g(D) = D^4 + D + 1$ utilizada na avaliação de desempenho do sistema de comunicação. A sexta forma de onda é a *entrada de erro* que é ativada (nível lógico alto) durante um pulso de clock simulando desta forma a ocorrência de erro durante a transmissão dos dados. A sétima forma de onda é a *saída do canal* $g(D) \oplus e$, e como um bit de erro foi inserido, a forma de onda na saída do canal terá uma inversão exatamente na posição onde o bit de erro foi inserido. A oitava forma de onda é um ponto de observação (teste 1). A nona forma de onda é a *saída do decodificador* $g^{-1}(e)$, que neste caso não é zero e sim uma função do erro inserido. A décima forma de onda é outro ponto de observação (teste 2). A décima primeira forma de onda é a *saída do registrador de erro*, que está indicando que um bit de erro foi inserido no canal. Ou seja, houve a detecção do erro na saída do circuito.

Este exemplo dado é somente uma das opções de utilização dos polinômios geradores de grau 4. Para os polinômios geradores de grau 4, ainda existem outras opções de equações. Todas as opções estão listadas abaixo:

$g(D) = D^4 + 1$

$g(D) = D^4 + D + 1$

$g(D) = D^4 + D^2 + 1$

$g(D) = D^4 + D^2 + D + 1$

$g(D) = D^4 + D^3 + 1$

$g(D) = D^4 + D^3 + D + 1$

$g(D) = D^4 + D^3 + D^2 + 1$

$g(D) = D^4 + D^3 + D^2 + D + 1$

A figura 4.19 ilustra a implementação de um esquema completo *de gerador, decodificador* e *registrador de erro* de forma que se implemente qualquer seqüência de dados usando uma das opções anteriores de equação (polinômios de grau 4):

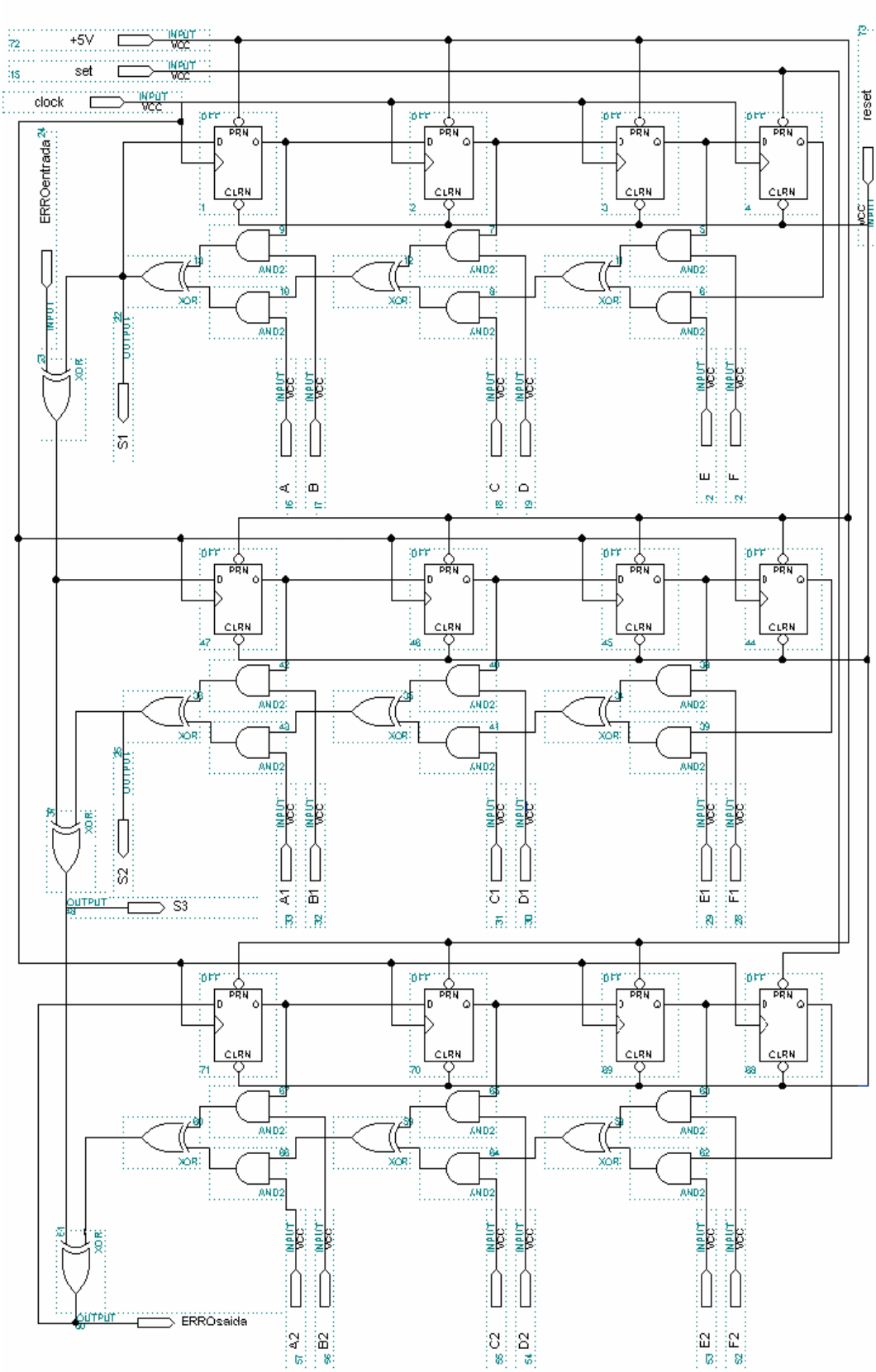


Figura 4.19 – Esquema completo (gerador, decodificador e registrador de erro) para utilização de polinômios de grau quatro.

O fluxograma da figura 4.20 ilustra o algoritmo de um esquema medidor de BER que utiliza polinômio de grau 4 como implementado pelo circuito da figura 4.19 e desenvolvido em linguagem AHDL (Anexo A):

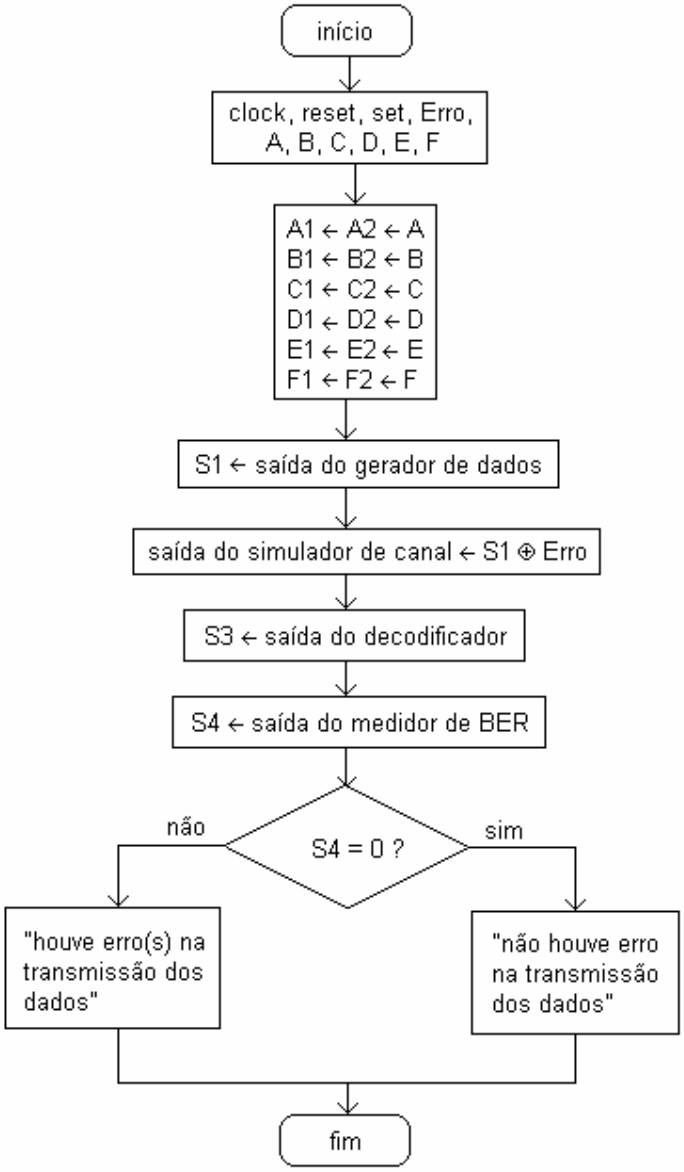


Figura 4.20 – Fluxograma para utilização de polinômios de grau 4 no medidor de BER

Como entrada de dados neste fluxograma tem-se: *clock, reset, set, erro, A, B, C, D, E* e *F,* onde *A1=A2=A, B1=B2=B, C1=C2=C, D1=D2=D, E1=E2=E* e *F1=F2=F.* A saída representada por *S1* recebe a saída do gerador de dados. A saída do simulador de canal recebe *S1* ou-exclusivo o *erro.* A saída representada por *S3* recebe a saída do decodificador. E a saída representada por *S4* recebe a saída do medidor de BER. Se a saída *S4* for igual a zero, implica

na não ocorrência de erro(s) durante a transmissão dos dados. Se a saída *S4* for diferente de zero, implica na ocorrência de erro(s) durante a transmissão dos dados.

Escolhendo implementar a seqüência resultante do polinômio $g(D) = D^4 + D + 1$, os valores de A, B, C, D, E e F que indicam os pontos de realimentação devem ser:

A = A1=A2=1

B = B1=B2=0

C = C1=C2=1

D = D1=D2=0

E = E1=E2=1

F = F1=F2=1.

A figura 4.21 ilustra as formas de onda para exemplo do polinômio $g(D) = D^4 + D + 1$, implementado através do circuito da figura 4.19 (circuito criado para implementar todas as equações dos polinômios de grau 4) com um bit de erro inserido no canal e detectado pelo registrador de erro:

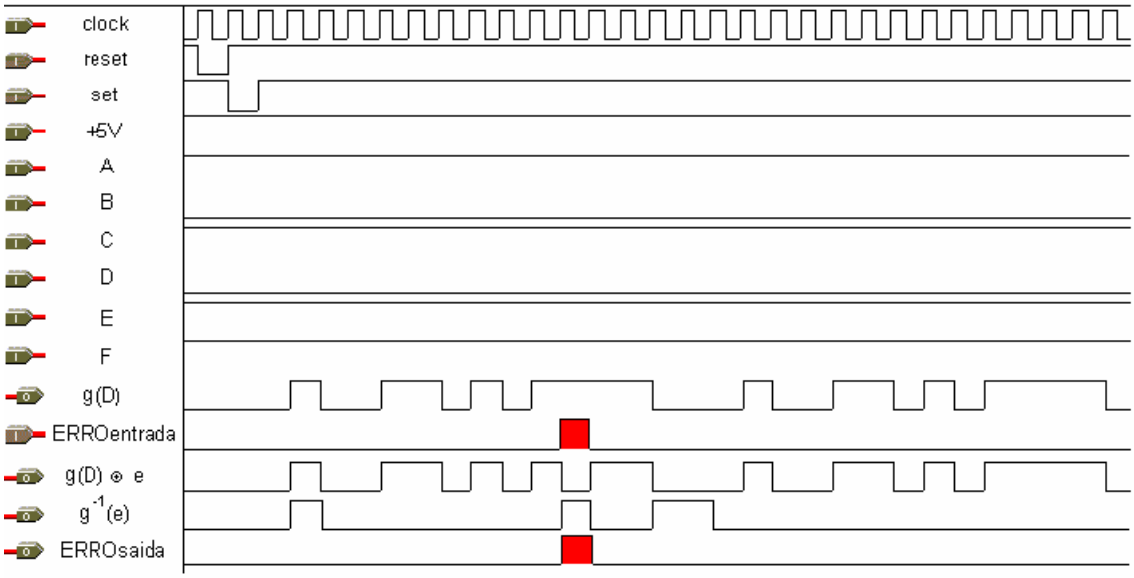


Figura 4.21 – Formas de onda de um sistema que utiliza polinômio gerador $g(D) = D^4 + D + 1$*, com inserção de um bit de erro, e os respectivos valores de A, B, C, D, E, e F*

A primeira forma de onda é o gerador de *clock*. A segunda forma de onda é a entrada *reset* que deve ser ativada (nível lógico baixo) somente no primeiro pulso de clock, entregando nível lógico baixo na saída de todos os flip-flops e permanecendo desativada (nível lógico alto) o tempo restante. A terceira forma de onda é a entrada *set* que deve ser

ativada (nível lógico baixo) somente no segundo pulso de clock, entregando nível lógico alto na saída do último flip-flop do bloco gerador de dados e do bloco registrador de erro. Nos demais pulsos de clock a entrada *set* permanece desativada (nível lógico alto). A quarta entrada é a *alimentação* fixada em *+5 Volts* (nível lógico alto) para permanecer desativada. A quinta, sexta, sétima, oitava, nona e décima ondas são as *entradas de realimentação do polinômio*. A décima primeira forma de onda, é *a saída do gerador* $g(D)$, ou seja, a seqüência de dados pseudo-aleatória gerada pelo polinômio $g(D) = D^4 + D + 1$ que é transmitida através do canal. A décima segunda forma de onda é a *entrada de erro* que é ativada (nível lógico alto) durante um pulso de clock simulando a ocorrência de erro durante a transmissão dos dados. A décima terceira forma de onda é a *saída do canal* $g(D) \oplus e$, e como um bit de erro foi inserido, a forma de onda na saída do canal terá uma inversão exatamente na posição onde o bit de erro foi inserido. A décima quarta forma de onda é a *saída do decodificador* $g^{-1}(e)$, que neste caso não é zero e sim uma função do erro inserido. Neste caso o erro inserido é triplicado. A décima quinta forma de onda é *a saída do registrador de erro*, que está indicando que um bit de erro foi inserido no canal. Ou seja, o erro que foi simulado na entrada do canal, está sendo detectado na saída do circuito.

Para os primeiros experimentos, foi utilizado a ferramenta "MAX+PLUS II software" [11, 12] da companhia ALTERA que permite a construção de circuitos e possui em sua estrutura um simulador para os circuitos projetados. Para o projeto em linguagem VHDL, também foi utilizado o software "MAX+PLUS II" e a ferramenta de simulação para o mesmo. Antes, porém, utilizou-se a linguagem AHDL (Altera Hardware Description Language) para o desenvolvimento de um projeto primário. O programa em AHDL para o circuito da figura 4.19 está listado no Anexo A.

Sempre o maior e o menor grau do polinômio escolhido devem existir na equação, para que a realimentação do circuito seja garantida.

No caso do exemplo da figura 4.19, D^4 e D^0(ou 1) sempre devem existir. Para que isso seja possível então;

A=A1=A2=1.

C=C1=C2=1.

E=E1=E2=1.

O programa em linguagem VHDL para implementação de um medidor de BER que utiliza qualquer polinômio de grau D^m, com $m \leq 32$ está listado no Anexo B. O programa principal (medidor 32) é composto por 3 blocos: o bloco 1 que é o gerador; o bloco 2 que é o

decodificador e o bloco 3 que é o registrador de erro. Além destes três blocos, existem os subprogramas para representarem os flip-flops tipo D com entrada 'reset' e os flip-flops tipo D com entradas 'set' e 'reset'.

O fluxograma da figura 4.22 ilustra o algoritmo para utilização de qualquer polinômio de grau D^m, com $m \leq 32$ como desenvolvido em linguagem VHDL no Anexo B:

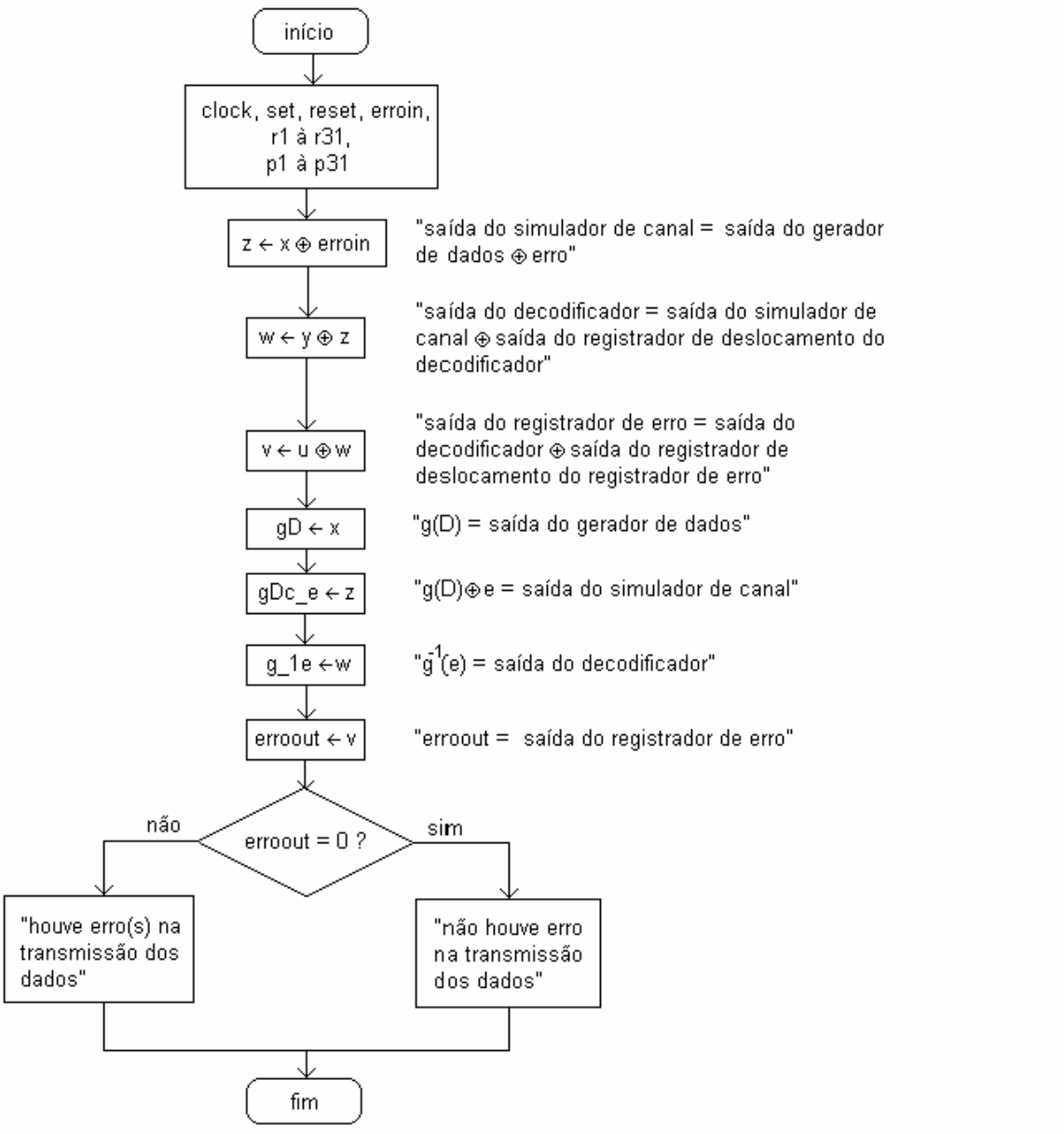


Figura 4.22 - Fluxograma para utilização de qualquer polinômio de grau D^m, com $m \leq 32$ no medidor de BER

Como entrada de dados neste fluxograma tem-se: *clock, reset, set, erroin, r1 à r31* e *p1 à p31* que são as entradas de controle de realimentação do circuito. A saída do simulador de

canal representada por *z* recebe a saída do *gerador de dados* representado por *x* ou-exclusivo o *erro simulado* representado por *erroin.* A saída do decodificador representada por *w* recebe a saída do *simulador de canal* representado por *y* ou exclusivo a saída do *registrador de deslocamento do decodificador* representado por *z*. A saída do registrador de erro representada por *v* recebe a saída do *decodificador* representado por *u* ou exclusivo a saída do *registrador de deslocamento do registrador de erro* representado por *w*. $g(D)$ recebe a *saída do gerador de dados x.* $g(D) \oplus e$ recebe a *saída do simulador de canal z.* $g^{-1}(e)$ recebe a *saída do decodificador w.* E o *erroout* recebe a *saída do registrador de erro v.* Se *erroout* for igual a zero, implica na não ocorrência de erro(s) durante a transmissão dos dados. Se *erroout* for diferente de zero, implica na ocorrência de erro(s) durante a transmissão dos dados.

É importante salientar que se a velocidade de operação do circuito implementado ultrapassar a velocidade de operação que o dispositivo (chip) suporta, este não será capaz de produzir a medida de taxa de erro de bits do sistema em que está sendo utilizado.

A freqüência de clock do circuito implementado também deve ser a freqüência que o dispositivo (chip) suporta, e se isso não acontecer, o circuito projetado não será capaz de produzir a medida de taxa de erro de bits do sistema em que está sendo utilizado.

Este projeto foi desenvolvido para detectar taxa de erro de bits de até 10^{-5}, que é a taxa máxima de erro "tolerada" na transmissão de áudio, para que se assegure a inteligibilidade da informação. Ou seja, pode ser detectado 1 bit de erro em até 100.000 bits transmitidos.

A implementação final deste projeto foi desenvolvida em linguagem de programação de hardware de alto nível VHDL devido às facilidades para futura implementação em dispositivo FPGA. Uma vez que a programação desenvolvida em software provido de simulação teve resultados concernentes ao proposto neste trabalho, considera-se que a implementação em dispositivo FPGA também traga resultados eficientes.

No capítulo 5 são relatadas as conclusões obtidas deste projeto e alguns trabalhos futuros são propostos.

CAPÍTULO 5 – CONCLUSÃO

Neste capítulo são explanados os resultados obtidos dos testes do projeto medidor de BER. Para trabalhos futuros são feitas sugestões de implementações.

Durante os desenvolvimentos feitos neste trabalho, obteve-se resultados parciais e globais exatamente como os esperados, conforme apresentados no capítulo 4. Foram analisados três blocos separadamente – gerador de dados; simulador de ruído no canal; decodificador e medidor de BER. Os resultados dos projetos de cada bloco foram concernentes aos esperados conforme discutido no capítulo 4. E no desenvolvimento do projeto global também foram obtidos os resultados esperados, ou seja, conseguiu-se detectar os erros ocorridos durante a transmissão de dados em um dado intervalo de tempo, obtendo uma medida de taxa de erro de bit em um sistema de comunicação digital.

A implementação deste projeto é simples e de baixo custo quando comparado aos dispositivos já existentes, pois utiliza dispositivos como flip-flops tipo D e lógica combinacional.

Propostas de Trabalhos Futuros

Algumas propostas de trabalhos futuros estão listadas abaixo:

1) Implementação de um medidor de BER que contenha: registro de contador de bit de 32 bits de comprimento, registro de contador de erro de bit de 32 bits de comprimento, inserção de bit de erro programável.
2) Implementação de um medidor de BER que possua transmissor e receptor totalmente independentes e porta de controle paralelo de 8 bits.
3) Implementação de um contador de bits errados para visualização em display de cristal líquido LCD conforme diagrama de blocos lógicos da figura 5.1 e levantar propriedades estatísticas dos polinômios propostos.

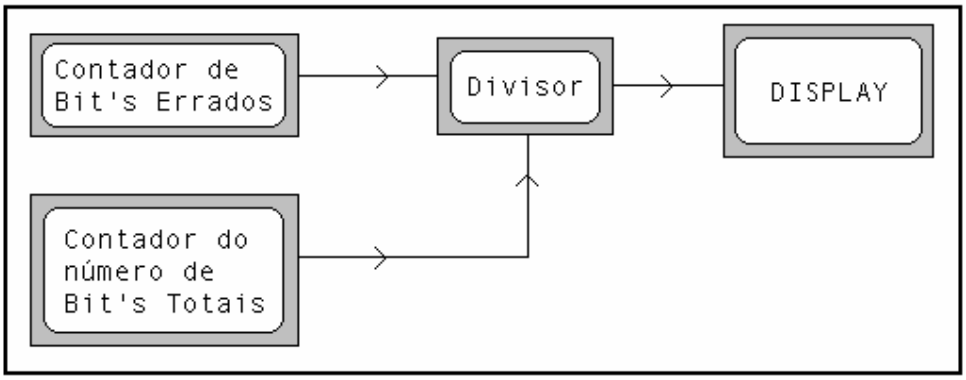


Figura 5.1 – Diagrama de blocos para visualização do número de bits errados em display de cristal líquido

Referências Bibliográficas

[1] Tomasi, Wayne: “Advanced Electronic Communications Systems”, Prentice-Hall International Editions, New Jersey, 1992.

[2] Haykin, Simon: “Digital Communications”, John Wiley & Sons, Canadá, 1988.

[3] Lin, Shu; Costello, Daniel J. Jr.: “Error Control Coding: Fundamentals and Applications”, Prentice-Hall, New Jersey, 1983.

[4] http://www.lcad.icmc.sc.usp.br/~jbatista/sce511/redes.doc

[5] Portelinha, Francisco M.: “Uma Técnica de Processamento Digital para MODEM QPSK”, Universidade Estadual de Campinas, Campinas, 1994.

[6] Newcombe, E.Alexander; Pasupathy, Subbarayan: “Error Rate Monitoring For Digital Communications”, Proceedings of the IEEE, vol 70, n° 8, agosto 1982.

[7] Pattavina, Jeffrey S.: “Bit Error Rate Estimation using CRC Codes”, July 17/2000.

[8] Schooley, Larry C.; McCurnin, Thomas W.: “Measurement of Bit Errors in Data Transmission Systems” IEEE transactions on communications, January 1976.

[9] MAX+PLUS II, Manuais CPLD’s MAX 3000 A e MAX, Altera Corporation, 1983.

[10] International Telecommunication Union ITU-T, O.150: “General requirements for instrumentation for performance measurements on digital transmission equipment”, Helsinki, March 1-12, 1992 (revisado em 1993 e 1996).

[11] MAX+PLUS II Software, Altera Corporation, 1983.

[12] MAX+PLUS II “User Guides”, Altera Corporation, 1983.

ANEXO A – Programa em linguagem AHDL de medidor de BER que utiliza polinômio gerador de grau 4 ou menor**.**

```
subdesign ber11c
(
clock          :      input; --gerador de clock--
reset          :      input; --entrada reset--
set            :      input; --entrada de set--
A,B,C,D,E,F    :      input; --entradas de realimentação do polinômio--
Erro           :      input; --entrada do gerador de erro--
S1, S2, S3, S4 :      output;--saídas do gerador, observação, decodificador e
                             registrador de erro respectivamente--
)

variable
d1      :      DFF;   --quarto flip-flop tipo D do gerador de dados--
d2      :      DFF;   --terceiro flip-flop tipo D do gerador de dados--
d3      :      DFF;   --segundo flip-flop tipo D do gerador de dados--
d4      :      DFF;   --primeiro flip-flop tipo D do gerador de dados--
d5      :      DFF;   --quarto flip-flop tipo D do decodificador--
d6      :      DFF;   --terceiro flip-flop tipo D do decodificador--
d7      :      DFF;   --segundo flip-flop tipo D do decodificador--
d8      :      DFF;   --primeiro flip-flop tipo D do decodificador--
d9      :      DFF;   --quarto flip-flop tipo D do registrador de erro--
d10     :      DFF;   --terceiro flip-flop tipo D do registrador de erro--
d11     :      DFF;   --segundo flip-flop tipo D do registrador de erro--
d12     :      DFF;   --primeiro flip-flop tipo D do registrador de erro--
no_d1   :      node;  --saída da porta ou exclusivo referente à D1 e D0 do
                        gerador--
no_d2   :      node;  --saída da porta ou exclusivo referente à D2, D1 e D0 do
                        gerador--
no_d3   :     node;   --saída da porta ou exclusivo referente à D3, D2, D1 e D0
                        do gerador--
no_d4   :      node;  --saída do canal--
no_d5   :      node;  --saída da porta ou exclusivo referente à D1 e D0 do
                        decodificador-
no_d6   :      node;  --saída da porta ou exclusivo referente à D2, D1 e D0 do
                        decodificador--
no_d7   :      node;  --saída da porta ou exclusivo referente à D3, D2, D1 e D0--
no_d8   :      node;  --saída do docodificador--
no_d9   :      node;  --saída da porta ou exclusivo referente à D1 e D0 do
                        registrador de erro--
no_d10:        node;  --saída da porta ou exclusivo referente à D2, D1 e D0 do
                        registrador erro--
no_d11:        node;  --saída da porta ou exclusivo referente à D3, D2, D1 e D0
                        registrador de erro--
no_d12:        node;  --saída do registrador de erro--

begin
no_d1 = F&d2.q $ E&d1.q; --saída do terceiro flip-flop "E" entrada F, ou
```

```
                          exclusivo, saída do quarto flip-flop "E" entrada E do
                          gerador--
no_d2 = D&d3.q $ C&no_d1; --saída do segundo flip-flop "E" entrada D, ou
                          exclusivo, saída do nó 1 "E" entrada C do gerador--
no_d3 = B&d4.q $ A&no_d2; --saída do primeiro flip-flop "E" entrada B, ou
                           exclusivo, saída do nó 2 "E" entrada A do gerador--
no_d4 = no_d3 $ Erro; --nó 3 ou exclusivo erro--
no_d5 = E&d5.q $ F&d6.q; --saída do terceiro flip-flop "E" entrada F, ou
                          exclusivo, saída do quarto flip-flop "E" entrada E do
                          decodificador--
 no_d6 = C&no_d5 $ D&d7.q; --saída do segundo flip-flop "E" entrada D, ou
                            exclusivo,saída do nó 5 "E" entrada C do
                            decodificador--
no_d7 = A&no_d6 $ B&d8.q; --saída do primeiro flip-flop "E" entrada B,ou
                          exclusivo,-- ---saída do nó 6 "E" entrada A do
                          decodificador--
no_d8 = no_d4 $ no_d7; --nó 4 ou exclusivo nó 7--
no_d9 = E&d9.q $ F&d10.q; --saída do terceiro flip-flop "E" entrada F, ou
                          exclusivo, saída do quarto flip-flop "E" entrada E do
                          registrador de erro--
no_d10 = C&no_d9 $ D&d11.q; --saída do segundo flip-flop "E" entrada D, ou
                             exclusivo, saída do nó 9 "E" entrada C do gerador-
no_d11 = A&no_d10 $ B&d12.q; --saída do primeiro flip-flop "E" entrada B, ou -
                              exclusivo, saída do nó 10 "E" entrada A do
                              gerador--
no_d12 = no_d11 $ no_d8;      --nó 11 ou exclusivo nó 8--
d12.d = no_d12;               --entrada do primeiro flip-flop do registrador de
                              erro recebe nó 12--
S1 = no_d3;                   --saída S1 = nó 3--
S2 = no_d7;                   --saída S2 = nó 7--
S3 = no_d8;                   --saída S3 = nó 8--
S4 = no_d12;                  --saída S4 = nó 12--
d4.clk= clock; --entrada clock do primeiro flip-flop do gerador recebe gerador de
                clock--
d4.prn  = VCC; --entrada preset do primeiro flip-flop do gerador recebe +5 volts-
d4.clrn = reset; --entrada clear do primeiro flip-flop do gerador recebe reset--
d4.d    = no_d3; --entrada D do primeiro flip-flop do gerador recebe nó 3--
d3.clk  = clock; --entrada clock do segundo flip-flop do gerador recebe gerador
                  de clock--
d3.prn  = VCC; --entrada preset do segundo flip-flop do gerador recebe +5 volts--
d3.clrn = reset; --entrada clear do segundo flip-flop do gerador recebe reset--
d3.d    = d4.q; --entrada D do segundo flip-flop do gerador recebe saída do
                primeiro flip-flop do gerador--
d2.clk  = clock; --entrada clock do terceiro flip-flop do gerador recebe gerador
                  de clock--
d2.prn  = VCC; --entrada preset do terceiro flip-flop do gerador recebe +5 volts-
d2.clrn = reset; --entrada clear do terceiro flip-flop do gerador recebe reset--
d2.d    = d3.q; --entrada D do terceiro flip-flop do gerador recebe saída do
                segundo flip-flop do gerador--
d1.clk  = clock; --entrada clock do quarto flip-flop do gerador recebe gerador de
                  clock--
```

```
d1.prn  = set; --entrada preset do quarto flip-flop do gerador recebe entrada
                 set--
d1.clrn = reset; --entrada clear do quarto flip-flop do gerador recebe reset--
d1.d    = d2.q; --entrada D do quarto flip-flop do gerador recebe saída do
                 terceiro flip-flop do gerador--
d8.clk  = clock; --entrada clock do primeiro flip-flop do decodificador recebe
                  gerador de clock--
d8.prn  = VCC; --entrada preset do primeiro flip-flop do decodificador  recebe +5
                 volts--
d8.clrn = reset; --entrada  clear do primeiro flip-flop do decodificador recebe
                   reset--
d8.d    = no_d4; --entrada D do primeiro flip-flop do decodificador recebe nó 4--
d7.clk  = clock; --entrada clock do segundo flip-flop do decodificador recebe
                   gerador de clock--
d7.prn  = VCC; --entrada preset do segundo flip-flop do decodificador recebe +5
                 volts--
d7.clrn = reset; --entrada  clear do segundo flip-flop do decodificador recebe
                   reset--
d7.d    = d8.q; --entrada D do segundo flip-flop do gerador recebe saída do
                 primeiro flip-flop do decodificador--
d6.clk  = clock; --entrada clock do terceiro flip-flop do decodificador recebe
                   gerador de clock--
d6.prn  = VCC; --entrada preset do terceiro flip-flop do decodificador recebe +5
                 volts--
d6.clrn = reset; --entrada  clear do terceiro flip-flop do decodificador recebe
                   reset--
d6.d    = d7.q; --entrada D do terceiro flip-flop do gerador recebe saída do
                 segundo flip-flop do decodificador--
d5.clk  = clock; --entrada clock do quarto flip-flop do decodificador recebe
                   gerador de clock--
d5.prn  = VCC; --entrada preset do quarto flip-flop do decodificador recebe +5
                 volts--
d5.clrn = reset; --entrada  clear do quarto flip-flop do decodificador recebe
                   reset--
d5.d    = d6.q; --entrada D do quarto flip-flop do decodificador recebe saída do
                  terceiro flip-flop do decodificador--
d12.clk= clock; --entrada clock do primeiro flip-flop do registrador de erro
                  recebe gerador de clock--
d12.prn= VCC; --entrada preset do primeiro flip-flop do registrador de erro
                recebe +5volts--
d12.clrn = reset; --entrada  clear do primeiro flip-flop do registrador de erro
                    recebe reset--
d12.d  = no_d12; --entrada D do primeiro flip-flop do registrador de erro recebe
                    nó 12--
d11.clk= clock; --entrada clock do segundo flip-flop do registrador de erro
                   recebe gerador de clock--
d11.prn= VCC; --entrada preset do segundo flip-flop do registrador de erro recebe
                 +5volts--
d11.clrn = reset; --entrada  clear do segundo flip-flop do registrador de erro
                     recebe reset--
d11.d = d12.q; --entrada D do segundo flip-flop do registrador de erro recebe
```

```
                saída do primeiro flip-flop do registrador de erro--
d10.clk= clock; --entrada clock do terceiro flip-flop do registrador de erro
                 recebe gerador de clock--
d10.prn= VCC; --entrada preset do terceiro flip-flop do registrador de erro
                  recebe +5volts--
d10.clrn = reset; --entrada  clear do terceiro flip-flop do registrador de erro
                    recebe reset--
d10.d = d11.q; --entrada D do terceiro flip-flop do registrador de erro recebe
                 saída do segundo flip-flop do registrador de erro--
d9.clk  = clock; --entrada clock do quarto flip-flop do registrador de erro
                  recebe gerador de clock--
d9.prn  = set; --entrada preset do quarto flip-flop do registrador de erro recebe
                entrada set--
d9.clrn = reset; --entrada  clear do terceiro flip-flop do registrador de erro
                  recebe reset--
d9.d = d10.q; --entrada D do quarto flip-flop do registrador de erro recebe saída
               do terceiro flip-flop do registrador de erro--
end;
```

ANEXO B – Programa em linguagem VHDL para medidor de BER que utiliza polinômio gerador de grau 32 ou menor.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY medidor32 IS

	PORT
	(
		clock                       : IN    STD_LOGIC;
		set                         : IN    STD_LOGIC;
		reset                       : IN    STD_LOGIC;
		erroin                      : IN    STD_LOGIC;
		r1,p1                       : IN    STD_LOGIC;
		r2,p2                       : IN    STD_LOGIC;
		r3,p3                       : IN    STD_LOGIC;
		r4,p4                       : IN    STD_LOGIC;
		r5,p5                       : IN    STD_LOGIC;
		r6,p6                       : IN    STD_LOGIC;
		r7,p7                       : IN    STD_LOGIC;
		p8,r8                       : IN    STD_LOGIC;
		p9,r9                       : IN    STD_LOGIC;
		p10,r10                     : IN    STD_LOGIC;
		p11,r11                     : IN    STD_LOGIC;
		p12,r12                     : IN    STD_LOGIC;
		p13,r13                     : IN    STD_LOGIC;
		p14,r14                     : IN    STD_LOGIC;
		p15,r15                     : IN    STD_LOGIC;
		p16,r16                     : IN    STD_LOGIC;
		p17,r17                     : IN    STD_LOGIC;
		p18,r18                     : IN    STD_LOGIC;
		p19,r19                     : IN    STD_LOGIC;
		p20,r20                     : IN    STD_LOGIC;
		p21,r21                     : IN    STD_LOGIC;
		p22,r22                     : IN    STD_LOGIC;
		p23,r23                     : IN    STD_LOGIC;
		p24,r24                     : IN    STD_LOGIC;
		p25,r25                     : IN    STD_LOGIC;
		p26,r26                     : IN    STD_LOGIC;
		p27,r27                     : IN    STD_LOGIC;
		p28,r28                     : IN    STD_LOGIC;
		p29,r29                     : IN    STD_LOGIC;
		p30,r30                     : IN    STD_LOGIC;
		p31,r31                     : IN    STD_LOGIC;
		gD,gDc_e,g_1e               : OUT STD_LOGIC;
		erroout                     : OUT STD_LOGIC
	);

END medidor32;
```

```
ARCHITECTURE a OF medidor32 IS

	SIGNAL  x,z,y,w,v,u		: STD_LOGIC;

	component bloco1medx32 port(
		clock				: IN	STD_LOGIC;
		reset				: IN	STD_LOGIC;
		set				: IN	STD_LOGIC;
		p1,r1				: IN	STD_LOGIC;
		p2,r2				: IN	STD_LOGIC;
		p3,r3				: IN	STD_LOGIC;
		p4,r4				: IN	STD_LOGIC;
		p5,r5				: IN	STD_LOGIC;
		p6,r6				: IN	STD_LOGIC;
		p7,r7				: IN	STD_LOGIC;
		p8,r8				: IN	STD_LOGIC;
		p9,r9				: IN	STD_LOGIC;
		p10,r10				: IN	STD_LOGIC;
		p11,r11				: IN	STD_LOGIC;
		p12,r12				: IN	STD_LOGIC;
		p13,r13				: IN	STD_LOGIC;
		p14,r14				: IN	STD_LOGIC;
		p15,r15				: IN	STD_LOGIC;
		p16,r16				: IN	STD_LOGIC;
		p17,r17				: IN	STD_LOGIC;
		p18,r18				: IN	STD_LOGIC;
		p19,r19				: IN	STD_LOGIC;
		p20,r20				: IN	STD_LOGIC;
		p21,r21				: IN	STD_LOGIC;
		p22,r22				: IN	STD_LOGIC;
		p23,r23				: IN	STD_LOGIC;
		p24,r24				: IN	STD_LOGIC;
		p25,r25				: IN	STD_LOGIC;
		p26,r26				: IN	STD_LOGIC;
		p27,r27				: IN	STD_LOGIC;
		p28,r28				: IN	STD_LOGIC;
		p29,r29				: IN	STD_LOGIC;
		p30,r30				: IN	STD_LOGIC;
		p31,r31				: IN	STD_LOGIC;
		s				: OUT STD_LOGIC
		);
	end component;

	component bloco2medx32 port(
		clock				: IN	STD_LOGIC;
		reset				: IN	STD_LOGIC;
		p1,r1				: IN	STD_LOGIC;
		p2,r2				: IN	STD_LOGIC;
		p3,r3				: IN	STD_LOGIC;
		p4,r4				: IN	STD_LOGIC;
		p5,r5				: IN	STD_LOGIC;
```

```
        p6,r6                           : IN    STD_LOGIC;
        p7,r7                           : IN    STD_LOGIC;
        p8,r8                           : IN    STD_LOGIC;
        p9,r9                           : IN    STD_LOGIC;
        p10,r10                         : IN    STD_LOGIC;
        p11,r11                         : IN    STD_LOGIC;
        p12,r12                         : IN    STD_LOGIC;
        p13,r13                         : IN    STD_LOGIC;
        p14,r14                         : IN    STD_LOGIC;
        p15,r15                         : IN    STD_LOGIC;
        p16,r16                         : IN    STD_LOGIC;
        p17,r17                         : IN    STD_LOGIC;
        p18,r18                         : IN    STD_LOGIC;
        p19,r19                         : IN    STD_LOGIC;
        p20,r20                         : IN    STD_LOGIC;
        p21,r21                         : IN    STD_LOGIC;
        p22,r22                         : IN    STD_LOGIC;
        p23,r23                         : IN    STD_LOGIC;
        p24,r24                         : IN    STD_LOGIC;
        p25,r25                         : IN    STD_LOGIC;
        p26,r26                         : IN    STD_LOGIC;
        p27,r27                         : IN    STD_LOGIC;
        p28,r28                         : IN    STD_LOGIC;
        p29,r29                         : IN    STD_LOGIC;
        p30,r30                         : IN    STD_LOGIC;
        p31,r31                         : IN    STD_LOGIC;
        ent                             : IN    STD_LOGIC;
        s                               : OUT STD_LOGIC
    );
    end component;

    component bloco3medx32 port(
        clock                           : IN    STD_LOGIC;
        reset                           : IN    STD_LOGIC;
        set                             : IN    STD_LOGIC;
        p1,r1                           : IN    STD_LOGIC;
        p2,r2                           : IN    STD_LOGIC;
        p3,r3                           : IN    STD_LOGIC;
        p4,r4                           : IN    STD_LOGIC;
        p5,r5                           : IN    STD_LOGIC;
        p6,r6                           : IN    STD_LOGIC;
        p7,r7                           : IN    STD_LOGIC;
        p8,r8                           : IN    STD_LOGIC;
        p9,r9                           : IN    STD_LOGIC;
        p10,r10                         : IN    STD_LOGIC;
        p11,r11                         : IN    STD_LOGIC;
        p12,r12                         : IN    STD_LOGIC;
        p13,r13                         : IN    STD_LOGIC;
        p14,r14                         : IN    STD_LOGIC;
        p15,r15                         : IN    STD_LOGIC;
        p16,r16                         : IN    STD_LOGIC;
```

```
               p17,r17                         : IN    STD_LOGIC;
               p18,r18                         : IN    STD_LOGIC;
               p19,r19                         : IN    STD_LOGIC;
               p20,r20                         : IN    STD_LOGIC;
               p21,r21                         : IN    STD_LOGIC;
               p22,r22                         : IN    STD_LOGIC;
               p23,r23                         : IN    STD_LOGIC;
               p24,r24                         : IN    STD_LOGIC;
               p25,r25                         : IN    STD_LOGIC;
               p26,r26                         : IN    STD_LOGIC;
               p27,r27                         : IN    STD_LOGIC;
               p28,r28                         : IN    STD_LOGIC;
               p29,r29                         : IN    STD_LOGIC;
               p30,r30                         : IN    STD_LOGIC;
               p31,r31                         : IN    STD_LOGIC;
               ent                             : IN    STD_LOGIC;
               s                               : OUT STD_LOGIC
       );
       end component;


BEGIN
u1: bloco1medx32 port map (clock,reset,set,p1,r1,p2,r2,
                                    p3,r3,p4,r4,p5,r5,p6,r6,p7,r7,
                                    p8,r8,p9,r9,p10,r10,p11,r11,
                                    p12,r12,p13,r13,p14,r14,p15,r15,
                                    p16,r16,p17,r17,p18,r18,p19,
                                    r19,p20,r20,p21,r21,p22,r22,
                                    p23,r23,p24,r24,p25,r25,p26,
                                    r26,p27,r27,p28,r28,p29,r29,
                                    p30,r30,p31,r31,x);
u2: bloco2medx32 port map (clock,reset,p1,r1,p2,r2,
                                    p3,r3,p4,r4,p5,r5,p6,r6,p7,r7,
                                    p8,r8,p9,r9,p10,r10,p11,r11,
                                    p12,r12,p13,r13,p14,r14,p15,r15,
                                    p16,r16,p17,r17,p18,r18,p19,
                                    r19,p20,r20,p21,r21,p22,r22,
                                    p23,r23,p24,r24,p25,r25,p26,
                                    r26,p27,r27,p28,r28,p29,r29,
                                    p30,r30,p31,r31,z,y);
u3: bloco3medx32 port map (clock,reset,set,p1,r1,p2,r2,
                                    p3,r3,p4,r4,p5,r5,p6,r6,p7,r7,
                                    p8,r8,p9,r9,p10,r10,p11,
                                    r11,p12,r12,p13,r13,p14,r14,p15,r15,
                                    p16,r16,p17,r17,p18,r18,
                                    p19,r19,p20,r20,p21,r21,p22,r22,
                                    p23,r23,p24,r24,p25,r25,
                                    p26,r26,p27,r27,p28,r28,p29,r29,
                                    p30,r30,p31,r31,v,u);

z <= x xor erroin;
```

```
w <= y xor z;
v <= u xor w;
gD <= x;
gDc_e <= z;
g_1e <= w;
erroout <= v;

END a;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bloco1medx32 IS

	PORT
	(
		clock				: IN	STD_LOGIC;
		reset				: IN	STD_LOGIC;
		set				: IN	STD_LOGIC;
		p1,r1				: IN	STD_LOGIC;
		p2,r2				: IN	STD_LOGIC;
		p3,r3				: IN	STD_LOGIC;
		p4,r4				: IN	STD_LOGIC;
		p5,r5				: IN	STD_LOGIC;
		p6,r6				: IN	STD_LOGIC;
		p7,r7				: IN	STD_LOGIC;
		p8,r8				: IN	STD_LOGIC;
		p9,r9				: IN	STD_LOGIC;
		p10,r10				: IN	STD_LOGIC;
		p11,r11				: IN	STD_LOGIC;
		p12,r12				: IN	STD_LOGIC;
		p13,r13				: IN	STD_LOGIC;
		p14,r14				: IN	STD_LOGIC;
		p15,r15				: IN	STD_LOGIC;
		p16,r16				: IN	STD_LOGIC;
		p17,r17				: IN	STD_LOGIC;
		p18,r18				: IN	STD_LOGIC;
		p19,r19				: IN	STD_LOGIC;
		p20,r20				: IN	STD_LOGIC;
		p21,r21				: IN	STD_LOGIC;
		p22,r22				: IN	STD_LOGIC;
		p23,r23				: IN	STD_LOGIC;
		p24,r24				: IN	STD_LOGIC;
		p25,r25				: IN	STD_LOGIC;
		p26,r26				: IN	STD_LOGIC;
		p27,r27				: IN	STD_LOGIC;
		p28,r28				: IN	STD_LOGIC;
		p29,r29				: IN	STD_LOGIC;
		p30,r30				: IN	STD_LOGIC;
		p31,r31				: IN	STD_LOGIC;
		s				: OUT	STD_LOGIC
	);

END bloco1medx32;

ARCHITECTURE a OF bloco1medx32 IS

	SIGNAL ctrsetx1,ctrsetx2,ctrsetx3,ctrsetx4,ctrsetx5,ctrsetx6,ctrsetx7,ctrsetx8,
	ctrsetx9,ctrsetx10,ctrsetx11,ctrsetx12,ctrsetx13,ctrsetx14,ctrsetx15,
	ctrsetx16,ctrsetx17,ctrsetx18,ctrsetx19,ctrsetx20,ctrsetx21,ctrsetx22,
```

```
        ctrsetx23,ctrsetx24,ctrsetx25,ctrsetx26,ctrsetx27,ctrsetx28,ctrsetx29,
        ctrsetx30,ctrsetx31,
        x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,
        x17,x18,x19,x20,x21,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,
        i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12,i13,i14,i15,i16,i17,i18,
        i19,i20,i21,i22,i23,i24,i25,i26,i27,i28,i29,i30,
        a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,ss,t,u,v,x,y,w,z,a1,b1,c1,d1,
        e1,f1,g1,h1,ii1,j1,k1,l1,m1,n1,o1,pp1,q1,rr1,s1,t1,uu1,v1,xx1,y1,w1,z1,
        a2,b2,c2,d2,e2,f2,g2,h2,ii2,j2 : STD_LOGIC;

        component ffdrs port( d, clk, s, r   : in STD_LOGIC;
                                    q              : out STD_LOGIC);
        end component;

BEGIN

u1: ffdrs port map (x1,clock,set,reset,x0);
u2: ffdrs port map (x2,clock,ctrsetx1,reset,x1);
u3: ffdrs port map (x3,clock,ctrsetx2,reset,x2);
u4: ffdrs port map (x4,clock,ctrsetx3,reset,x3);
u5: ffdrs port map (x5,clock,ctrsetx4,reset,x4);
u6: ffdrs port map (x6,clock,ctrsetx5,reset,x5);
u7: ffdrs port map (x7,clock,ctrsetx6,reset,x6);
u8: ffdrs port map (x8,clock,ctrsetx7,reset,x7);
u9: ffdrs port map (x9,clock,ctrsetx8,reset,x8);
u10: ffdrs port map (x10,clock,ctrsetx9,reset,x9);
u11: ffdrs port map (x11,clock,ctrsetx10,reset,x10);
u12: ffdrs port map (x12,clock,ctrsetx11,reset,x11);
u13: ffdrs port map (x13,clock,ctrsetx12,reset,x12);
u14: ffdrs port map (x14,clock,ctrsetx13,reset,x13);
u15: ffdrs port map (x15,clock,ctrsetx14,reset,x14);
u16: ffdrs port map (x16,clock,ctrsetx15,reset,x15);
u17: ffdrs port map (x17,clock,ctrsetx16,reset,x16);
u18: ffdrs port map (x18,clock,ctrsetx17,reset,x17);
u19: ffdrs port map (x19,clock,ctrsetx18,reset,x18);
u20: ffdrs port map (x20,clock,ctrsetx19,reset,x19);
u21: ffdrs port map (x21,clock,ctrsetx20,reset,x20);
u22: ffdrs port map (x22,clock,ctrsetx21,reset,x21);
u23: ffdrs port map (x23,clock,ctrsetx22,reset,x22);
u24: ffdrs port map (x24,clock,ctrsetx23,reset,x23);
u25: ffdrs port map (x25,clock,ctrsetx24,reset,x24);
u26: ffdrs port map (x26,clock,ctrsetx25,reset,x25);
u27: ffdrs port map (x27,clock,ctrsetx26,reset,x26);
u28: ffdrs port map (x28,clock,ctrsetx27,reset,x27);
u29: ffdrs port map (x29,clock,ctrsetx28,reset,x28);
u30: ffdrs port map (x30,clock,ctrsetx29,reset,x29);
u31: ffdrs port map (x31,clock,ctrsetx30,reset,x30);
u32: ffdrs port map (x32,clock,ctrsetx31,reset,x31);

ctrsetx1 <= set or p1;
ctrsetx2 <= set or p2;
```

```
ctrsetx3 <= set or p3;
ctrsetx4 <= set or p4;
ctrsetx5 <= set or p5;
ctrsetx6 <= set or p6;
ctrsetx7 <= set or p7;
ctrsetx8 <= set or p8;
ctrsetx9 <= set or p9;
ctrsetx10 <= set or p10;
ctrsetx11 <= set or p11;
ctrsetx12 <= set or p12;
ctrsetx13 <= set or p13;
ctrsetx14 <= set or p14;
ctrsetx15 <= set or p15;
ctrsetx16 <= set or p16;
ctrsetx17 <= set or p17;
ctrsetx18 <= set or p18;
ctrsetx19 <= set or p19;
ctrsetx20 <= set or p20;
ctrsetx21 <= set or p21;
ctrsetx22 <= set or p22;
ctrsetx23 <= set or p23;
ctrsetx24 <= set or p24;
ctrsetx25 <= set or p25;
ctrsetx26 <= set or p26;
ctrsetx27 <= set or p27;
ctrsetx28 <= set or p28;
ctrsetx29 <= set or p29;
ctrsetx30 <= set or p30;
ctrsetx31 <= set or p31;

a <= x1 and r1;
b <= x0 and p1;
i1 <= a xor b;
c <= x2 and r2;
d <= i1 and p2;
i2 <= c xor d;
e <= x3 and r3;
f <= i2 and p3;
i3 <= e xor f;
g <= x4 and r4;
h <= i3 and p4;
i4 <= g xor h;
i <= x5 and r5;
j <= i4 and p5;
i5 <= i xor j;
k <= x6 and r6;
l <= i5 and p6;
i6 <= k xor l;
m <= x7 and r7;
n <= i6 and p7;
i7 <= m xor n;
```

```
o <= x8 and r8;
p <= i7 and p8;
i8 <= o xor p;
q <= x9 and r9;
r <= i8 and p9;
i9 <= q xor r;
ss <= x10 and r10;
t <= i9 and p10;
i10 <= ss xor t;
u <= x11 and r11;
v <= i10 and p11;
i11 <= u xor v;
x <= x12 and r12;
y <= i11 and p12;
i12 <= x xor y;
w <= x13 and r13;
z <= i12 and p13;
i13 <= w xor z;
a1 <= x14 and r14;
b1 <= i13 and p14;
i14 <= a1 xor b1;
c1 <= x15 and r15;
d1 <= i14 and p15;
i15<= c1 xor d1;
e1 <= x16 and r16;
f1 <= i15 and p16;
i16 <= e1 xor f1;
g1 <= x17 and r17;
h1 <= i16 and p17;
i17 <= g1 xor h1;
ii1 <= x18 and r18;
j1 <= i17 and p18;
i18 <= ii1 xor j1;
k1 <= x19 and r19;
l1 <= i18 and p19;
i19 <= k1 xor l1;
m1 <= x20 and r20;
n1 <= i19 and p20;
i20 <= m1 xor n1;
o1 <= x21 and r21;
pp1 <= i20 and p21;
i21 <= o1 xor pp1;
q1 <= x22 and r22;
rr1 <= i21 and p22;
i22 <= q1 xor rr1;
s1 <= x23 and r23;
t1 <= i22 and p23;
i23 <= s1 xor t1;
uu1 <= x24 and r24;
v1 <= i23 and p24;
i24 <= uu1 xor v1;
```

```
xx1 <= x25 and r25;
y1 <= i24 and p25;
i25 <= xx1 xor y1;
w1 <= x26 and r26;
z1 <= i25 and p26;
i26 <= w1 xor z1;
a2 <= x27 and r27;
b2 <= i26 and p27;
i27 <= a2 xor b2;
c2 <= x28 and r28;
d2 <= i27 and p28;
i28 <= c2 xor d2;
e2 <= x29 and r29;
f2 <= i28 and p29;
i29 <= e2 xor f2;
g2 <= x30 and r30;
h2 <= i29 and p30;
i30 <= g2 xor h2;
ii2 <= x31 and r31;
j2 <= i30 and p31;
x32 <= ii2 xor j2;
s <= x32;
END a;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bloco2medx32 IS

	PORT
	(
		clock				: IN	STD_LOGIC;
		reset				: IN	STD_LOGIC;
		p1,r1				: IN	STD_LOGIC;
		p2,r2				: IN	STD_LOGIC;
		p3,r3				: IN	STD_LOGIC;
		p4,r4				: IN	STD_LOGIC;
		p5,r5				: IN	STD_LOGIC;
		p6,r6				: IN	STD_LOGIC;
		p7,r7				: IN	STD_LOGIC;
		p8,r8				: IN	STD_LOGIC;
		p9,r9				: IN	STD_LOGIC;
		p10,r10				: IN	STD_LOGIC;
		p11,r11				: IN	STD_LOGIC;
		p12,r12				: IN	STD_LOGIC;
		p13,r13				: IN	STD_LOGIC;
		p14,r14				: IN	STD_LOGIC;
		p15,r15				: IN	STD_LOGIC;
		p16,r16				: IN	STD_LOGIC;
		p17,r17				: IN	STD_LOGIC;
		p18,r18				: IN	STD_LOGIC;
		p19,r19				: IN	STD_LOGIC;
		p20,r20				: IN	STD_LOGIC;
		p21,r21				: IN	STD_LOGIC;
		p22,r22				: IN	STD_LOGIC;
		p23,r23				: IN	STD_LOGIC;
		p24,r24				: IN	STD_LOGIC;
		p25,r25				: IN	STD_LOGIC;
		p26,r26				: IN	STD_LOGIC;
		p27,r27				: IN	STD_LOGIC;
		p28,r28				: IN	STD_LOGIC;
		p29,r29				: IN	STD_LOGIC;
		p30,r30				: IN	STD_LOGIC;
		p31,r31				: IN	STD_LOGIC;
		ent				: IN	STD_LOGIC;
		s				: OUT	STD_LOGIC
	);

END bloco2medx32;

ARCHITECTURE a OF bloco2medx32 IS

	SIGNAL
		x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,
		x17,x18,x19,x20,x21,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,
```

```
                i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12,i13,i14,i15,i16,i17,i18,
                i19,i20,i21,i22,i23,i24,i25,i26,i27,i28,i29,i30,
                a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,ss,t,u,v,x,y,w,z,a1,b1,c1,d1,
                e1,f1,g1,h1,ii1,j1,k1,l1,m1,n1,o1,pp1,q1,rr1,s1,t1,uu1,
                v1,xx1,y1,w1,z1,
                a2,b2,c2,d2,e2,f2,g2,h2,ii2,j2 : STD_LOGIC;

        component ffdr port(  d, clk,  r     : in STD_LOGIC;
                              q                    : out STD_LOGIC);
        end component;

BEGIN

u1: ffdr port map (x1,clock,reset,x0);
u2: ffdr port map (x2,clock,reset,x1);
u3: ffdr port map (x3,clock,reset,x2);
u4: ffdr port map (x4,clock,reset,x3);
u5: ffdr port map (x5,clock,reset,x4);
u6: ffdr port map (x6,clock,reset,x5);
u7: ffdr port map (x7,clock,reset,x6);
u8: ffdr port map (x8,clock,reset,x7);
u9: ffdr port map (x9,clock,reset,x8);
u10: ffdr port map (x10,clock,reset,x9);
u11: ffdr port map (x11,clock,reset,x10);
u12: ffdr port map (x12,clock,reset,x11);
u13: ffdr port map (x13,clock,reset,x12);
u14: ffdr port map (x14,clock,reset,x13);
u15: ffdr port map (x15,clock,reset,x14);
u16: ffdr port map (x16,clock,reset,x15);
u17: ffdr port map (x17,clock,reset,x16);
u18: ffdr port map (x18,clock,reset,x17);
u19: ffdr port map (x19,clock,reset,x18);
u20: ffdr port map (x20,clock,reset,x19);
u21: ffdr port map (x21,clock,reset,x20);
u22: ffdr port map (x22,clock,reset,x21);
u23: ffdr port map (x23,clock,reset,x22);
u24: ffdr port map (x24,clock,reset,x23);
u25: ffdr port map (x25,clock,reset,x24);
u26: ffdr port map (x26,clock,reset,x25);
u27: ffdr port map (x27,clock,reset,x26);
u28: ffdr port map (x28,clock,reset,x27);
u29: ffdr port map (x29,clock,reset,x28);
u30: ffdr port map (x30,clock,reset,x29);
u31: ffdr port map (x31,clock,reset,x30);
u32: ffdr port map (ent,clock,reset,x31);

a <= x1 and r1;
b <= x0 and p1;
i1 <= a xor b;
c <= x2 and r2;
d <= i1 and p2;
```

```
i2 <= c xor d;
e <= x3 and r3;
f <= i2 and p3;
i3 <= e xor f;
g <= x4 and r4;
h <= i3 and p4;
i4 <= g xor h;
i <= x5 and r5;
j <= i4 and p5;
i5 <= i xor j;
k <= x6 and r6;
l <= i5 and p6;
i6 <= k xor l;
m <= x7 and r7;
n <= i6 and p7;
i7 <= m xor n;
o <= x8 and r8;
p <= i7 and p8;
i8 <= o xor p;
q <= x9 and r9;
r <= i8 and p9;
i9 <= q xor r;
ss <= x10 and r10;
t <= i9 and p10;
i10 <= ss xor t;
u <= x11 and r11;
v <= i10 and p11;
i11 <= u xor v;
x <= x12 and r12;
y <= i11 and p12;
i12 <= x xor y;
w <= x13 and r13;
z <= i12 and p13;
i13 <= w xor z;
a1 <= x14 and r14;
b1 <= i13 and p14;
i14 <= a1 xor b1;
c1 <= x15 and r15;
d1 <= i14 and p15;
i15<= c1 xor d1;
e1 <= x16 and r16;
f1 <= i15 and p16;
i16 <= e1 xor f1;
g1 <= x17 and r17;
h1 <= i16 and p17;
i17 <= g1 xor h1;
ii1 <= x18 and r18;
j1 <= i17 and p18;
i18 <= ii1 xor j1;
k1 <= x19 and r19;
l1 <= i18 and p19;
```

```
i19 <= k1 xor l1;
m1 <= x20 and r20;
n1 <= i19 and p20;
i20 <= m1 xor n1;
o1 <= x21 and r21;
pp1 <= i20 and p21;
i21 <= o1 xor pp1;
q1 <= x22 and r22;
rr1 <= i21 and p22;
i22 <= q1 xor rr1;
s1 <= x23 and r23;
t1 <= i22 and p23;
i23 <= s1 xor t1;
uu1 <= x24 and r24;
v1 <= i23 and p24;
i24 <= uu1 xor v1;
xx1 <= x25 and r25;
y1 <= i24 and p25;
i25 <= xx1 xor y1;
w1 <= x26 and r26;
z1 <= i25 and p26;
i26 <= w1 xor z1;
a2 <= x27 and r27;
b2 <= i26 and p27;
i27 <= a2 xor b2;
c2 <= x28 and r28;
d2 <= i27 and p28;
i28 <= c2 xor d2;
e2 <= x29 and r29;
f2 <= i28 and p29;
i29 <= e2 xor f2;
g2 <= x30 and r30;
h2 <= i29 and p30;
i30 <= g2 xor h2;
ii2 <= x31 and r31;
j2 <= i30 and p31;
s <= ii2 xor j2;
END a;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bloco3medx32 IS

	PORT
	(
		clock				: IN	STD_LOGIC;
		reset				: IN	STD_LOGIC;
		set				: IN	STD_LOGIC;
		p1,r1				: IN	STD_LOGIC;
		p2,r2				: IN	STD_LOGIC;
		p3,r3				: IN	STD_LOGIC;
		p4,r4				: IN	STD_LOGIC;
		p5,r5				: IN	STD_LOGIC;
		p6,r6				: IN	STD_LOGIC;
		p7,r7				: IN	STD_LOGIC;
		p8,r8				: IN	STD_LOGIC;
		p9,r9				: IN	STD_LOGIC;
		p10,r10				: IN	STD_LOGIC;
		p11,r11				: IN	STD_LOGIC;
		p12,r12				: IN	STD_LOGIC;
		p13,r13				: IN	STD_LOGIC;
		p14,r14				: IN	STD_LOGIC;
		p15,r15				: IN	STD_LOGIC;
		p16,r16				: IN	STD_LOGIC;
		p17,r17				: IN	STD_LOGIC;
		p18,r18				: IN	STD_LOGIC;
		p19,r19				: IN	STD_LOGIC;
		p20,r20				: IN	STD_LOGIC;
		p21,r21				: IN	STD_LOGIC;
		p22,r22				: IN	STD_LOGIC;
		p23,r23				: IN	STD_LOGIC;
		p24,r24				: IN	STD_LOGIC;
		p25,r25				: IN	STD_LOGIC;
		p26,r26				: IN	STD_LOGIC;
		p27,r27				: IN	STD_LOGIC;
		p28,r28				: IN	STD_LOGIC;
		p29,r29				: IN	STD_LOGIC;
		p30,r30				: IN	STD_LOGIC;
		p31,r31				: IN	STD_LOGIC;
		ent				: IN	STD_LOGIC;
		s				: OUT	STD_LOGIC
	);

END bloco3medx32;

ARCHITECTURE a OF bloco3medx32 IS

	SIGNAL ctrsetx1,ctrsetx2,ctrsetx3,ctrsetx4,ctrsetx5,ctrsetx6,ctrsetx7,ctrsetx8,
	ctrsetx9,ctrsetx10,ctrsetx11,ctrsetx12,ctrsetx13,ctrsetx14,ctrsetx15,
```

```
        ctrsetx16,ctrsetx17,ctrsetx18,ctrsetx19,ctrsetx20,ctrsetx21,ctrsetx22,
        ctrsetx23,ctrsetx24,ctrsetx25,ctrsetx26,ctrsetx27,ctrsetx28,ctrsetx29,
        ctrsetx30,ctrsetx31,
        x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,
        x17,x18,x19,x20,x21,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,
        i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12,i13,i14,i15,i16,i17,i18,
        i19,i20,i21,i22,i23,i24,i25,i26,i27,i28,i29,i30,
        a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,ss,t,u,v,x,y,w,z,a1,b1,c1,d1,
        e1,f1,g1,h1,ii1,j1,k1,l1,m1,n1,o1,pp1,q1,rr1,s1,t1,uu1,v1,xx1,y1,w1,z1,
        a2,b2,c2,d2,e2,f2,g2,h2,ii2,j2 : STD_LOGIC;

        component ffdrs port( d, clk, s, r   : in STD_LOGIC;
                              q                    : out STD_LOGIC);
        end component;

BEGIN

u1: ffdrs port map (x1,clock,set,reset,x0);
u2: ffdrs port map (x2,clock,ctrsetx1,reset,x1);
u3: ffdrs port map (x3,clock,ctrsetx2,reset,x2);
u4: ffdrs port map (x4,clock,ctrsetx3,reset,x3);
u5: ffdrs port map (x5,clock,ctrsetx4,reset,x4);
u6: ffdrs port map (x6,clock,ctrsetx5,reset,x5);
u7: ffdrs port map (x7,clock,ctrsetx6,reset,x6);
u8: ffdrs port map (x8,clock,ctrsetx7,reset,x7);
u9: ffdrs port map (x9,clock,ctrsetx8,reset,x8);
u10: ffdrs port map (x10,clock,ctrsetx9,reset,x9);
u11: ffdrs port map (x11,clock,ctrsetx10,reset,x10);
u12: ffdrs port map (x12,clock,ctrsetx11,reset,x11);
u13: ffdrs port map (x13,clock,ctrsetx12,reset,x12);
u14: ffdrs port map (x14,clock,ctrsetx13,reset,x13);
u15: ffdrs port map (x15,clock,ctrsetx14,reset,x14);
u16: ffdrs port map (x16,clock,ctrsetx15,reset,x15);
u17: ffdrs port map (x17,clock,ctrsetx16,reset,x16);
u18: ffdrs port map (x18,clock,ctrsetx17,reset,x17);
u19: ffdrs port map (x19,clock,ctrsetx18,reset,x18);
u20: ffdrs port map (x20,clock,ctrsetx19,reset,x19);
u21: ffdrs port map (x21,clock,ctrsetx20,reset,x20);
u22: ffdrs port map (x22,clock,ctrsetx21,reset,x21);
u23: ffdrs port map (x23,clock,ctrsetx22,reset,x22);
u24: ffdrs port map (x24,clock,ctrsetx23,reset,x23);
u25: ffdrs port map (x25,clock,ctrsetx24,reset,x24);
u26: ffdrs port map (x26,clock,ctrsetx25,reset,x25);
u27: ffdrs port map (x27,clock,ctrsetx26,reset,x26);
u28: ffdrs port map (x28,clock,ctrsetx27,reset,x27);
u29: ffdrs port map (x29,clock,ctrsetx28,reset,x28);
u30: ffdrs port map (x30,clock,ctrsetx29,reset,x29);
u31: ffdrs port map (x31,clock,ctrsetx30,reset,x30);
u32: ffdrs port map (ent,clock,ctrsetx31,reset,x31);

ctrsetx1 <= set or p1;
```

```
ctrsetx2 <= set or p2;
ctrsetx3 <= set or p3;
ctrsetx4 <= set or p4;
ctrsetx5 <= set or p5;
ctrsetx6 <= set or p6;
ctrsetx7 <= set or p7;
ctrsetx8 <= set or p8;
ctrsetx9 <= set or p9;
ctrsetx10 <= set or p10;
ctrsetx11 <= set or p11;
ctrsetx12 <= set or p12;
ctrsetx13 <= set or p13;
ctrsetx14 <= set or p14;
ctrsetx15 <= set or p15;
ctrsetx16 <= set or p16;
ctrsetx17 <= set or p17;
ctrsetx18 <= set or p18;
ctrsetx19 <= set or p19;
ctrsetx20 <= set or p20;
ctrsetx21 <= set or p21;
ctrsetx22 <= set or p22;
ctrsetx23 <= set or p23;
ctrsetx24 <= set or p24;
ctrsetx25 <= set or p25;
ctrsetx26 <= set or p26;
ctrsetx27 <= set or p27;
ctrsetx28 <= set or p28;
ctrsetx29 <= set or p29;
ctrsetx30 <= set or p30;
ctrsetx31 <= set or p31;

a <= x1 and r1;
b <= x0 and p1;
i1 <= a xor b;
c <= x2 and r2;
d <= i1 and p2;
i2 <= c xor d;
e <= x3 and r3;
f <= i2 and p3;
i3 <= e xor f;
g <= x4 and r4;
h <= i3 and p4;
i4 <= g xor h;
i <= x5 and r5;
j <= i4 and p5;
i5 <= i xor j;
k <= x6 and r6;
l <= i5 and p6;
i6 <= k xor l;
m <= x7 and r7;
n <= i6 and p7;
```

```
i7 <= m xor n;
o <= x8 and r8;
p <= i7 and p8;
i8 <= o xor p;
q <= x9 and r9;
r <= i8 and p9;
i9 <= q xor r;
ss <= x10 and r10;
t <= i9 and p10;
i10 <= ss xor t;
u <= x11 and r11;
v <= i10 and p11;
i11 <= u xor v;
x <= x12 and r12;
y <= i11 and p12;
i12 <= x xor y;
w <= x13 and r13;
z <= i12 and p13;
i13 <= w xor z;
a1 <= x14 and r14;
b1 <= i13 and p14;
i14 <= a1 xor b1;
c1 <= x15 and r15;
d1 <= i14 and p15;
i15<= c1 xor d1;
e1 <= x16 and r16;
f1 <= i15 and p16;
i16 <= e1 xor f1;
g1 <= x17 and r17;
h1 <= i16 and p17;
i17 <= g1 xor h1;
ii1 <= x18 and r18;
j1 <= i17 and p18;
i18 <= ii1 xor j1;
k1 <= x19 and r19;
l1 <= i18 and p19;
i19 <= k1 xor l1;
m1 <= x20 and r20;
n1 <= i19 and p20;
i20 <= m1 xor n1;
o1 <= x21 and r21;
pp1 <= i20 and p21;
i21 <= o1 xor pp1;
q1 <= x22 and r22;
rr1 <= i21 and p22;
i22 <= q1 xor rr1;
s1 <= x23 and r23;
t1 <= i22 and p23;
i23 <= s1 xor t1;
uu1 <= x24 and r24;
v1 <= i23 and p24;
```

```
i24 <= uu1 xor v1;
xx1 <= x25 and r25;
y1 <= i24 and p25;
i25 <= xx1 xor y1;
w1 <= x26 and r26;
z1 <= i25 and p26;
i26 <= w1 xor z1;
a2 <= x27 and r27;
b2 <= i26 and p27;
i27 <= a2 xor b2;
c2 <= x28 and r28;
d2 <= i27 and p28;
i28 <= c2 xor d2;
e2 <= x29 and r29;
f2 <= i28 and p29;
i29 <= e2 xor f2;
g2 <= x30 and r30;
h2 <= i29 and p30;
i30 <= g2 xor h2;
ii2 <= x31 and r31;
j2 <= i30 and p31;
x32 <= ii2 xor j2;
s <= x32;
END a;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ffdr is
    port (
        d              : in STD_LOGIC;
        clk            : in STD_LOGIC;
        r              : in STD_LOGIC;
        q              : out STD_LOGIC
    );
end ffdr;

architecture ffdr_arch of ffdr is
begin
  -- <<enter your statements here>>
  ffdr: process (d,clk,r)
  begin
       if r='0'
        then q <= '0';
        elsif rising_edge(clk)
        then q <= d;
       end if;
  end process ffdr;
end ffdr_arch;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ffdrs is
    port (
        d              : in STD_LOGIC;
        clk            : in STD_LOGIC;
        s              : in STD_LOGIC;
        r              : in STD_LOGIC;
        q              : out STD_LOGIC
    );
end ffdrs;

architecture ffdr_arch of ffdrs is
begin
  -- <<enter your statements here>>
  ffdrs: process (d,clk,s,r)
  begin
       if s='0'
        then q <= '1';
        elsif r='0'
              then q <='0';
        elsif rising_edge(clk)
        then q <= d;
       end if;
  end process ffdrs;
end ffdr_arch;
```