DESENVOLVIMENTO DE UM "KIT" DIDÁTICO PARA APRENDIZAGEM DA TÉCNICA DE CODIFICAÇÃO REED SOLOMON

Samir Shehady[1]; Arnaldo Megrich[2]

1 Aluno de Iniciação Científica da Escola de Engenharia Mauá (EEM/CEUN-IMT);
2 Professor da Escola de Engenharia Mauá (EEM/CEUN-IMT).

Resumo. *O objetivo do trabalho consiste em desenvolver, através de CPLD's (Complex Programmable Logic Device), o "hardware" e o "software" associado de modo a criar um sistema transmissor - meio de propagação - receptor, destinado a facilitar a compreensão e o estudo da técnica de correção de erros conhecida como Reed Solomon.*

Introdução

Quando do tratamento de mensagens (sinais), dois tipos de problemas podem ser definidos: a degradação da capacidade de armazenagem ou transmissão destas mensagens e ruído no processo transmissivo, ou seja, o acoplamento (randômico) de erros nas mensagens. Um código corretor de erros tem por objetivo a recuperação de mensagens que, no meio de transmissão, tenham sido submetidos a ruídos. Todo sistema de envio de mensagens possui, no mínimo, um dentre um conjunto de processos corretivos de erros possíveis. Como exemplo, nas técnicas de Televisão Digital é implementado o código Reed Solomon.

O código Reed Solomon (RS) foi inventado em 1960 por Irving S. Reed e Gustave Solomon, que eram membros do MIT Lincoln Laboratory. Esses códigos constituem uma subclasse de uma ampla classe de códigos cíclicos denominada de Códigos BCH (Bose - Chaudhuri - Hocquenghem) e encontram-se entre os códigos mais poderosos no que diz respeito à capacidade de correção de erros, sendo largamente utilizados em muitos sistemas digitais em funções tais como o armazenamento de dados (CD, DVD, Códigos de Barras, etc), transmissão de dados, codificação de correio eletrônico, transmissão via satélite, Televisão Digital, modems de alta velocidade (ADSL), comunicações Wireless e móveis (celulares, ligações utilizando microondas), entre outros.

A figura abaixo (Figura 1) mostra a estrutura do código Reed Solomon.

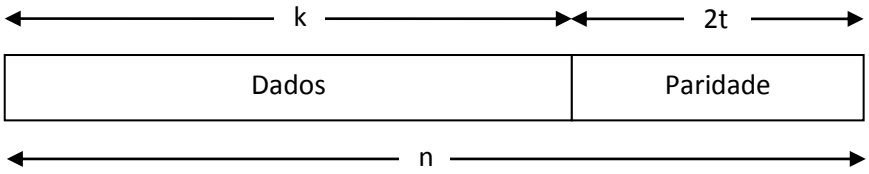

Figura 1 – Estrutura associada ao código Reed Solomon.

No que concerne a essa formatação, *n* corresponde ao comprimento do bloco, *k* à parcela de dados efetivos, *n-k = 2t* representa os símbolos de paridade, sendo *t* a quantidade máxima de símbolos com erros que podem ser corrigidos. O número de bits por símbolo é designado por *m*. A representação genérica de uma codificação RS é definida por RS(n,k) e está descrita na literatura (Megrich (2009), Lin e Costello (1983), Morelos-Zaragoza (2006), Wicker e Bhargava (1994)).

O código Reed Solomon é capaz de corrigir qualquer combinação composta por *t* erros (ou menos) em que *t* é dado por [(n-k)/2], segundo as notações acima. Na simbologia adotada, [x] indica o maior número inteiro que não excede x.

O codificador Reed Solomon pode, a partir da ocorrência de erros (devido a ruídos, interferências, etc) e do processamento de cada bloco, tentar corrigir os referidos erros e recuperar os dados originais. A correção consiste em um procedimento que opera através da

super amostragem de um polinômio construído a partir dos dados (polinômio auxiliar). Este polinômio é avaliado em um conjunto de pontos, sendo estes valores enviados. Considerando-se que o polinômio em questão esta sendo amostrado mais freqüentemente que o necessário, se muitos pontos forem constantemente recebidos, o receptor terá condições de identificar as características do polinômio original mesmo que pontos degradados tenham sido tratados. A programação de codificação seguiu os princípios da formatação sistemática dos dados, sendo esta uma das alternativas possíveis na técnica Reed Solomon e usualmente a mais conveniente por permitir a separação dos sinais codificados dos de mensagem.

Quanto à decodificação, a técnica Reed Solomon pode corrigir erros e efetuar apagamentos. Um apagamento pode ocorrer quando a posição de um símbolo errôneo é conhecida. Um decodificador pode corrigir até t erros ou, alternativamente, executar até $2t$ apagamentos. Quando uma seqüência de informações é decodificada, três tipos de ocorrências podem ter lugar, a saber: a) se acontecer $2m + r < 2t$ (ou seja, m erros e r apagamentos), a seqüência de informações originais será corretamente recuperada; b) o decodificador detecta que não é capaz de recuperar a seqüência de informações original e indica a impossibilidade; c) o decodificador interpreta as cadeias binárias de modo incorreto, recuperando a seqüência de informações degenerada, sem qualquer indicação a respeito. As rotinas de decodificação basearam-se no algoritmo iterativo de Berlekamp, descrito na literatura (Lin e Costello, 1983).

Material e Métodos

Através do uso de algoritmos especialmente desenvolvidos foi possível criar um código fonte em C++ referente à técnica de correção de erros Reed Solomon. Por meio deste código foi possível realizar simulações envolvendo diversas cadeias de informações binárias. Com o objetivo de desenvolver através de CPLD's o "hardware" e o "software" para operar o processamento Reed Solomon, a codificação em C++ foi transformada em código VHDL. Utilizando o "software" Symphony EDA – VHDL Simili foi possível desenvolver simulações e realizar testes semelhantes aqueles aqui apresentados. Foi possível variar a quantidade de dados bem como a quantidade de erros que podem ser corrigidos além da extensão dos dados, dentre outros parâmetros. Foi criado um procedimento denominado de Test Bench, o qual permite operacionalizar o procedimento por meio de inserção de cadeias binárias juntamente com a monitoração das respostas.

Em se tratando de construção do "hardware", utilizamos o programa Altera – Quartus II. O "software" Quartus II é um sistema de desenvolvimento voltado para o uso de CPLD's e FPGA's completamente integrado, fornecendo suporte para todas as necessidades de projeto: entrada de dados, síntese, alocação e roteamento do componente, simulação, análise de "timing" e programação de CPLD's / FPGA's. O projeto em questão procura criar três "hardwares" diferentes. O primeiro é destinado à inserção dos dados de entrada, além da geração do Campo Finito (Campo de Galois, procedimento este inerente ao processamento RS, o qual pode ser entendido como um conjunto de elementos que podem ser aplicados a uma operação aritmética básica com a ressalva de que qualquer das operações implica em valores contidos neste campo), a geração do polinômio auxiliar e por fim a codificação destes dados. Já o segundo "hardware" exibe e possibilita a alteração total ou parcial dos dados, simulando-se assim perturbações nas informações, semelhantes a intempéries. O terceiro "hardware" executa a decodificação dos dados, recuperando ou não a série original, em função da quantidade de erros estabelecidos. O diagrama de blocos do "kit" didático para a aprendizagem de técnicas de codificação Reed Solomon (Figura 2) exemplifica o processo aqui descrito.

Figura 2 – Diagrama de blocos do "kit" didático para a aprendizagem de técnicas de codificação Reed Solomon.

Os "hardwares" foram implementados em placas educacionais UP2 da família MAX7000S, a qual possui alta velocidade, baixo custo, memória EEPROM, densidade de 600 a 5000 portas utilizáveis, permitindo que as saídas sejam individualmente configuradas como dreno aberto dentre outras características.

A descrição em VHDL corresponde a um dos primeiros níveis na hierarquia do projeto de circuitos digitais. Esta descrição pode ser feita segundo duas filosofias: a comportamental e a estrutural.

A descrição comportamental do sistema é vista como uma "caixa negra" que implementa uma determinada funcionalidade. Essa funcionalidade foi escrita em VHDL com um conjunto de instruções, como seleção "if-then-else", ciclos "for" e "while", que são perfeitamente válidas em termos de programa mas pode, eventualmente, ser impossível de sintetizar num circuito.

Na descrição estrutural, o sistema é visto como uma interligação de componentes menos complexos que, por sua vez, formam blocos construídos a partir de circuitos ainda mais básicos, por exemplo um conversor do valor em hexadecimal para mostrar os dados nos displays, sendo construídos com um certo número de "flip-flops" (células de memória) e algumas portas lógicas. As memórias são construídas a partir de portas lógicas básicas.

Inicialmente foram desenvolvidos algoritmos para o primeiro "hardware", o qual recebe os dados de entrada, gera o campo finito (Galois), cria um polinômio auxiliar, codifica os dados gerando dados de paridade e por fim envia os dados para o segundo "hardware". Para tanto foi necessário criar uma ENTITY, a qual é a parte principal do projeto, por ser a interface do sistema que descreve as entradas e saídas. Também foi necessário criar vários PROCESS, os quais permitem descrever o comportamento do sistema ao longo do tempo, utilizando comandos seqüenciais. Foi criada também a ARCHITECTURE, que é o corpo do sistema, onde são feitas as atribuições, operações e comparações, dentre outras tarefas. Foi assim gerado um bloco designado por ENCODER. Estando desenvolvido o bloco, foi projetado o esquema elétrico contendo as entradas e saídas propostas, o qual é composto pelo bloco ENCODER e os devidos INPUTS e OUTPUTS.

Seguindo-se à criação do projeto, o mesmo deve ser compilado. Durante a compilação, a qual é realizada em várias etapas, erros de edição são verificados e arquivos auxiliares são gerados permitindo tanto a simulação do projeto quando a gravação do mesmo em dispositivo programável (placa UP2). Foram constatados diversos problemas na compilação com a programação, tais como a quantidade de bits a serem utilizados para a representação de números inteiros, a quantidade necessária de pinos e o estouro de memória, dentre outros. Foram realizados diversos procedimentos corretivos destes erros através da inserção de mais procedimentos, alteração nas rotinas e sub-rotinas. Todavia, não foi possível alcançar uma compilação correta.

Optou-se, então pela utilização da família de CPLD's FLEX10K, a qual permite o uso de uma grande quantidade de registradores e densidade de 10000 a 250000 portas utilizáveis, dentre outras facilidades. Com essa família de CPLD's pôde-se minimizar os erros constatados, porém, ainda não se atingiu uma compilação satisfatória.

Resultados e Discussão

Para as simulações foi utilizado o arquivo em VHDL e simulado no programa VHDL Simili. Foram fixados os três parâmetros abaixo, em todos os casos analisados:

Tamanho da palavra código (n) = 15
Quantidade de "bits" por palavra (para a geração do Campo de Galois) = 4
Dados efetivos (k) = 9

PARTE I

Para a PARTE I da simulação procurou-se manter para todos os casos os dados efetivos abaixo, assim como os dados recebidos do codificador:

Mensagem a ser transmitida (dados efetivos):

8	6	8	1	2	4	15	9	9

O codificador recebe os dados efetivos além dos dados de paridade.

15	13	10	0	6	13	8	6	8	1	2	4	15	9	9

Simulação 1

Número de erros que podem ser corrigidos (t) = 1

Antes da decodificação parte dos dados serão modificados. No caso, introduziu-se um distúrbio (indicado em negrito):

15	13	10	0	6	13	8	6	**7**	1	2	4	15	9	9

Ao injetar a seqüência agregada ao distúrbio no bloco decodificador, chegou-se à seguinte cadeia binária:

15	13	10	0	6	13	8	6	8	1	2	4	15	9	9

sabendo-se que os seis primeiros dados consistem nos dados de paridade, sendo os demais informações efetivamente impostas. Neste caso há a recuperação da seqüência original, pois a quantidade de distúrbios é menor ou igual a quantidade de erros que podem ser corrigidos (ou seja, um único).

Já com a introdução de dois distúrbios (também em negrito) tem-se:

0	13	10	0	6	13	8	6	8	1	2	4	15	9	**0**

Ao injetar a seqüência agregada ao distúrbio no bloco decodificador, chegou-se à seguinte cadeia binária:

0	13	10	0	6	13	8	6	8	1	2	4	15	9	0

Pôde-se verificar que não houve a recuperação dos dados originais, pois a quantidade de distúrbios é maior que a quantidade de erros que podem ser corrigidos.

Simulação 2

Número de erros que podem ser corrigidos (t) = 2

Antes da decodificação parte dos dados serão modificados. No caso, foram introduzidos dois distúrbios (em negrito):

15	13	10	**7**	6	13	8	6	8	1	2	4	15	**11**	9

Ao injetar a seqüência agregada ao distúrbio no bloco decodificador, chegou-se à seguinte cadeia binária:

15	13	10	0	6	13	8	6	8	1	2	4	15	9	9

Neste caso há recuperação da seqüência original, pois a quantidade de distúrbios é menor ou igual a quantidade de erros que podem ser corrigidos (ou seja, dois).

Já com a introdução de três distúrbios (em negrito) tem-se:

15	**14**	**1**	0	6	**0**	8	**10**	8	1	2	**3**	15	**8**	9

Ao injetar a seqüência agregada aos distúrbios no bloco decodificador, chegou-se à seguinte cadeia binária:

15	14	1	0	6	0	8	10	8	1	2	3	15	8	9

Pôde-se verificar que não houve a recuperação dos dados originais, pois a quantidade de distúrbios é maior do que a quantidade de erros que podem ser corrigidos.

Simulação 3

Número de erros que podem ser corrigidos (t) = 3

Antes da decodificação parte dos dados serão modificados. No caso, foram introduzidos três distúrbios (em negrito):

15	13	10	0	6	**1**	**15**	**14**	8	1	2	4	15	9	9

Ao injetar a seqüência agregada ao distúrbio no bloco decodificador, chegou-se à seguinte cadeia binária:

15	13	10	0	6	13	8	6	8	1	2	4	15	9	9

Neste caso ocorre a recuperação da seqüência original, pois a quantidade de distúrbios é menor ou igual a quantidade de erros que podem ser corrigidos (ou seja, três).

Já com a introdução de, por exemplo, cinco distúrbios (em negrito) tem-se:

0	13	**8**	0	6	13	**4**	6	8	1	**13**	4	15	9	**10**

Ao injetar a seqüência agregada ao distúrbio no bloco decodificador, chegou-se à seguinte cadeia binária:

0	13	8	0	6	13	4	6	8	1	13	4	15	9	10

Pôde-se verificar que não houve a recuperação dos dados originais, pois a quantidade de distúrbios é maior do que a quantidade de erros que podem ser corrigidos.

Simulação 4

Número de erros que podem ser corrigidos (t) = 4

Antes da decodificação parte dos dados serão modificados. No caso, foram introduzidos quatro distúrbios (em negrito):

15	13	**6**	**4**	6	13	8	6	**3**	1	2	**12**	15	9	9

Ao injetar a seqüência agregada ao distúrbio no bloco decodificador, chegou-se à seguinte cadeia binária:

15	13	6	4	6	13	8	6	3	1	2	12	15	9	9

O programa desenvolvido permite a correção de, no máximo, três erros, daí a inconsistência deste resultado. Sob tais circunstâncias o algoritmo falha e as informações originais são perdidas.

PARTE II

Para a segunda parte das simulações variam-se os dados efetivos.

Simulação 1

Mensagem a ser transmitida (dados efetivos):

0	0	0	0	0	0	0	0	0

O codificador recebe os dados efetivos além dos dados de paridade.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Número de erros que podem ser corrigidos (t) = 3

Antes da decodificação parte dos dados serão modificados. No caso, foram introduzidos três distúrbios (em negrito):

0	0	0	**8**	0	0	0	0	0	**11**	0	0	0	**4**	0

Ao injetar a seqüência agregada aos distúrbios no bloco decodificador, chegou-se à seguinte cadeia binária:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Neste caso há a recuperação da seqüência original, pois a quantidade de distúrbios é menor ou igual à quantidade de erros que podem ser corrigidos (ou seja, três). Pode-se

concluir a partir das simulações da PARTE I que se forem inseridos mais que três distúrbios não haverá correção devido às limitações da programação desenvolvida.

Simulação 2

Mensagem a ser transmitida (dados efetivos):

1	1	1	1	1	1	1	1	1

O codificador recebe os dados efetivos além dos dados de paridade.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Número de erros que podem ser corrigidos (t) = 2

Antes da decodificação parte dos dados serão modificados. No caso, foram inseridos dois distúrbios (em negrito):

12	1	1	1	1	1	1	1	1	1	**5**	1	1	1	1

Ao injetar a seqüência agregada aos distúrbios no bloco decodificador, chegou-se à seguinte cadeia binária:

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Neste caso há a recuperação da seqüência original, pois a quantidade de distúrbios é menor ou igual a quantidade de erros que podem ser corrigidos (ou seja, dois). Pode-se concluir a partir das simulações da PARTE I que se forem introduzidos mais que três distúrbios não haverá correção.

Conclusões

Pôde-se verificar que o código fonte, da forma que foi criado para a simulação, não permite uma implementação correta no ambiente Quartus II. A questão básica é que tal código não é compatível com o tipo de compilação e simulação que o programa executa (análise e síntese de projetos em VHDL). Uma das exigências freqüentemente feitas a uma descrição VHDL é a de que esta seja sintetizável, ou seja, o modelo descrito será convertido para estruturas de dados representando as conexões, blocos, componentes e portas lógicas. Isto acontece sempre que se pretende não somente simular e testar um circuito, como também fabricá-lo. A simulação no "software" Quartus II verifica se a função lógica e/ou seqüencial do projeto proposto e compilado segue os resultados esperados, o que não ocorreu em nosso caso. Mesmo sem a efetiva realização do "hardware", a funcionalidade do procedimento foi alcançada, o que pode ser constatado por meio das simulações executadas. A busca de soluções para a superação destas dificuldades está, no entanto, em andamento.

Referências Bibliográficas

Lin, S.; Costello, D.J. (1983) *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, New Jersey. Prentice-Hall.

Megrich, A. (2009) *Televisão Digital – Princípios e Técnicas*. São Paulo. Erica.

Morelos-Zaragoza R.H. (2006) *The Art of Error Correcting Coding.* 2nd Edition. Chichester, John Wiley & Sons Ltd.

Wicker, S.B.; Bhargava, V.K. (1994). *Reed-Solomon Codes and Their Applications.* Piscataway, New Jersey. Willey-IEEE Press.