Ponto de Controle 2
Implementação do PLHEADER Decoder para o receptor DVB-S2X

Leonardo Amorim de Araújo - 15/0039921
Email: leonardoaraujodf@gmail.com
Bitbucket: https://bitbucket.org/leonardoaraujodf/
Universidade de Brasília
St. Leste Projeção A – Gama Leste, Brasília – DF, 72444 – 240

***Resumo*—Este documento apresenta uma proposta de projeto final para a disciplina de Projeto de Circuitos Reconfiguráveis onde será implementado um bloco que realiza a identificação do PLHEADER do transmissor DVB-S2x.**

***Keywords*—DVB-S2X, Hadamard, FPGA, basys3**

I. Introdução

O DVB - *Digital Video Broadcasting*, ou também chamado de televisão digital, é um consórcio normativo universal. Seu objetivo é concordar especificações para sistemas de entrega de mídia digital, incluindo a transmissão. É uma iniciativa aberta do setor privado com uma taxa de associação anual, regida por um Memorando de Entendimento (MdE). [2].

O padrão DVB-S2 (EN 302 307) define a modulação de segunda geração e o sistema de codificação de canais para TV via satélite para utilizar as melhorias que surgiram desde a publicação do padrão DVB-S. O DVB-S2 é um padrão único e altamente flexível que cobre uma variedade de aplicações por satélite [1].

O DVB-S2 é o próximo passo lógico no desenvolvimento contínuo do DVB-S. Os métodos de codificação de canal inovadores e mais eficientes combinados com os modos de modulação de ordem superior permitem que os operadores transmitam até 30% mais dados ao usar o DVB-S2 em comparação com o DVB-S na mesma largura de banda do transponder e EIRP.

O sistema foi otimizado para os serviços de transmissão digital multicanal de televisão e os serviços de transmissão de televisão em alta definição (HDTV) a serem usados para a distribuição primária e secundária nas bandas do serviço via satélite fixo (FSS) e do serviço de transmissão via satélite (BSS).

A. PL Header Decoder

O bloco **PLHEADER** do receptor **DVB-S2X** recebe uma parte do **PLFRAME**, o desembaralha e o decodifica para fornecer em sua saída variáveis que indicam a taxa de código, eficiência de espectro, tamanho de código e presença ou não de códigos pilotos no sinal a ser recebido após o PLHEADER. Este bloco irá indicar aos outros blocos do receptor do DVB-S2X, portanto, o que esperar do **FECFRAME** [3].

Segundo a norma, o PLHEADER que tem um total de 90 símbolos, é composto por dois campos, o primeiro chamado de **SOF** - Start of Frame, que possui 26 símbolos, e identifica o começo do FRAME, e o **PLS CODE** - Physical Layer Signalling, que possui 64 símbolos [3].

O PLS CODE de 64 símbolos é gerado pela codificação de uma palavra de 7 bits através de um gerador de PLS CODE, que pode ser visualizado na Figura 1.

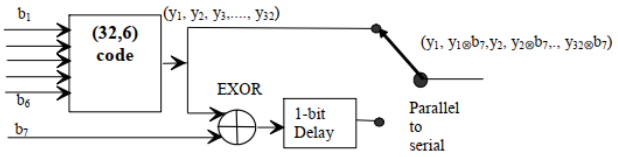

Figura 1. Gerador do PLS CODE

Este bloco utiliza uma matriz de **Reed-Muller** para gerar um frame de 32-bits, matriz esta que é mostrada na Figura 2. A sequência obtida recebe redundâncias entre cada bit de acordo com o bit menos significativo do PLS Code (b_7), fazendo o XOR com o bit atual e adicionado-o a frente, gerando uma sequência de 64-bits. Essa sequência de 64 bits é posteriormente embaralhada realizando-se o XOR com a sequência mostrada na Figura 3. Após todas as operações de codificação, a sequência recebe em sua parte mais significativa o número padrão SOF (0x18D2E82), resultando em uma frame PLHEADER com comprimento de 90-bits (mais detalhes disponíveis na norma DVB-S2X).

No recebimento após demodulação do PLS CODE, a palavra de 64 bits recebida estará provavelmente alterada devido ao ruído inerente ao canal de comunicação. Para que se possa saber qual foi a palavra que teve maior probabilidade ter sido enviada, deve-se realizar a correlação entre os 64 bits recebidos e os todas as sequências de 64 bits possíveis de serem enviadas pelo transmissor conforme a norma DVB-S2X.

Como os 7 bits geram 128 possibilidades de PLS CODE a serem enviados, seriam necessários 128 cálculos de correlação, gerando um custo computacional enorme. Uma alternativa para este problema é utilizar o método da Transformada Rápida de Hadamard [4], que fornece um vetor de números que indicam a

$$\mathbf{G} = \begin{bmatrix} 10010000101011000010110111011101 \\ 01010101010101010101010101010101 \\ 00110011001100110011001100110011 \\ 00001111000011110000111100001111 \\ 00000000111111110000000011111111 \\ 00000000000000001111111111111111 \\ 11111111111111111111111111111111 \end{bmatrix}$$

Figura 2. Matriz Reed-Muller

01110001100111011000001111001001010100110100001000101101111110101

Figura 3. Sequência de 64 bits para gerar o embaralhamento

probabilidade de cada header ter sido transmitido, porém com a vantagem de esta poder ser calculada utilizando somente somas e subtrações.

Além do fato a seguir, cada bit enviado está representado por um número complexo resultante da modulação da constelação BPSK Pi/2, com estes valores representados em 10 bits, onde o bit mais significativo refere-se ao sinal, os dois bits a seguir referem-se a parte inteira, e os 7 bits restantes referem-se a parte fracionária. Portanto, a representação dos símbolos é feita por **ponto fixo**.

Desta forma, este trabalho estará focado em uma solução para a escolha do header com a maior probabilidade de ter sido enviado com base no header recebido na entrada e dos blocos que realizam a transformada de hadamard[4].

II. Objetivos

Implementação de um bloco que verifique a probabilidade do header recebido ser um dos headers possíveis gerados pelo bloco PLS CODE do transmissor DVB-S2X.

III. Arquitetura de Hadware

A. FIFO

A arquitetura de hardware proposta consistiu primeiramente em definir uma FIFO - *First In First Out* de 64 posições, onde cada posição possui 10 bits. A Figura 4 mostra o diagrama de blocos criado. A FIFO possui 64 registradores de 10 bits cascateados, que são acionados ao mesmo tempo pela entrada **fifo_next_in** e um contador que conta de 0 a 63, e quando chega no número 63 aciona a saída **data_ready_out** alertando próximo estágio de que um array de 64 símbolos está disponível na saída. As entradas e saídas tem as seguintes funções:

Entradas:

- **fifo_next_in** (1 bit): Quando '1', permite que na próxima borda de subida de clock o valor no registrador n seja transferido para o registrador n+1, e que o valor em fifo_data_in seja enviado para o registrador 0. Também permite que o contador seja incrementado, se sua contagem for menor que 64.
- **fifo_reset_in** (1 bit): Quando '1', torna assincronamente todas as saídas dos registradores em 0 e zera o contador.
- **clk_in** (1 bit): Entrada de clock do componente.
- **fifo_data_in** (10 bits): Entrada de dados de 10 bits.

Saídas:

- **data_ready_out** (1 bit): Indica que 64 símbolos na saída estão disponíveis para serem lidos.
- **fifo_data_out** (64 x 10 bits): Saída de dados que possui 64 valores, cada um com 10 bits.

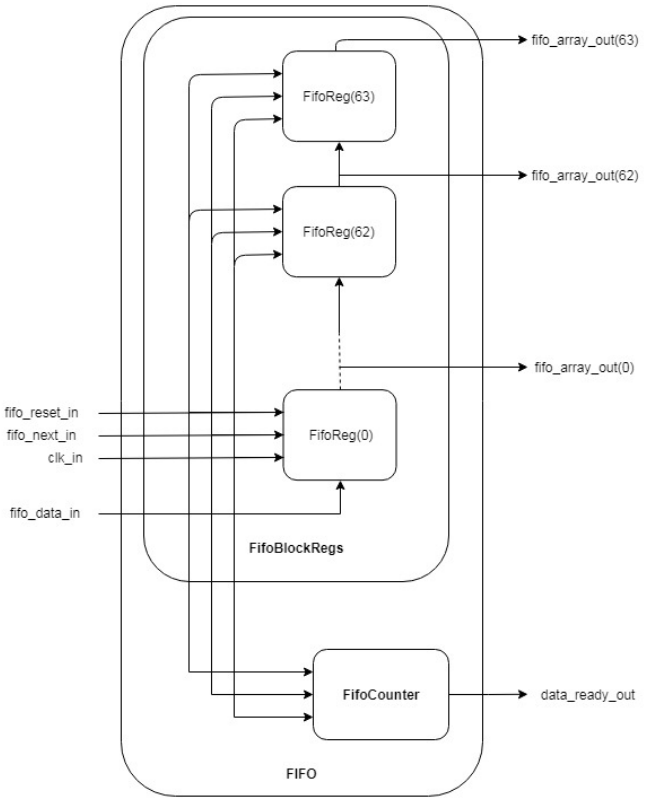

Figura 4. Diagrama de Blocos da *First In First Out*

B. Signal Changer

Foi criado também um bloco chamado de **SignalChanger** que recebe um dado de 10 bits, e calcula ou não, conforme o modo de operação recebido, o complemento de 2 deste dado. Além disso, o dado de entrada, que possui 3 bits de parte inteira, recebe um incremento de mais 6 bits de parte inteira, resultando portanto no dado de saída de 16 bits. O motivo será explicado logo mais adiante no bloco **SimpleAdder**. Para exemplificar, suponha que o dado na entrada seja 0110000000, e que peça-se para não realizar o complemento de 2. Então o dado de saída será 0000000110000000. Se fosse pedido para realizar o complemento de 2, então o dado seria 1111111010000000 (a parte fracionária não muda). O bloco SignalChanger é mostrado na Figura 5. As entradas e saídas são especificadas a seguir:

Entradas:

- **enable_in** (1 bit): Quando enable_in = ’1’, permite que o dado na entrada num_in seja calculado e enviado para a saída após na borda de subida clock.
- **reset_in** (1 bit): Torna o valor em num_out 0 assincronamente.
- **num_in** (10 bits): Dado de entrada de 10 bits.
- **clk_in** (1 bit): Clock de entrada.

Saídas:

- **num_out** (10 bits): Dado de saída de 16 bits.
- **ready_out** (1 bit): Indica que há um dado novo disponível na saída.

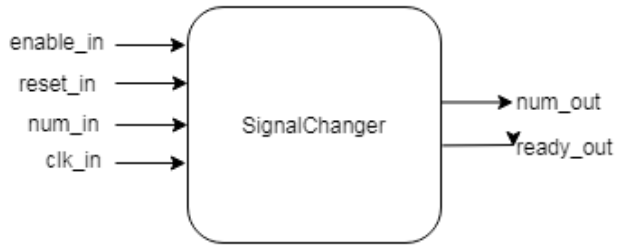

Figura 5. Bloco SignalChanger

Um bloco de 64 SignalChanger, chamado de BlockSignalChanger será ligado posteriormente à FIFO, recebendo 64 símbolos e selecionando, conforme o número enviado para o seletor de operação, se o respectivo símbolo será enviado para os somadores como complemento de 2 ou não. A associação destes blocos em um único bloco pode ser visualizado na Figura 6.

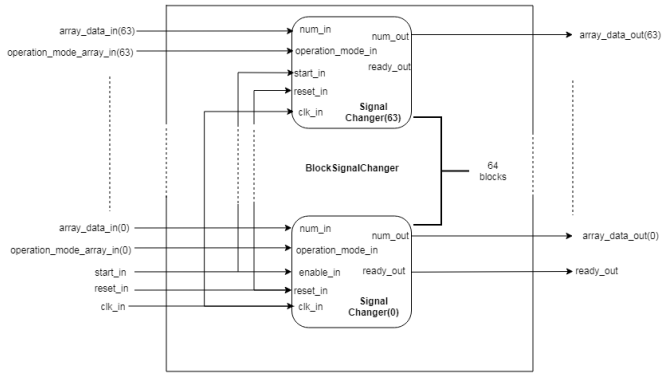

Figura 6. 64 SignalChanger em paralelo, bloco denominado BlockSignalChanger

As entradas e saídas deste bloco são especificadas a seguir:
Entradas:

- **array_data_in** (64 x 10 bits): um array de 64 posições, cada posição com comprimento de 10 bits. Estas entradas estão ligadas diretamente à saída da FIFO.
- **operation_mode_in** (64 bits): Um vetor de 64 bits que indica se o símbolo recebido na posição n deve ser enviado à saída n como complemento de 2 da entrada ou não.
- **start_in** (1 bit): Quando start_in = 1, os dados na entrada array_data_in serão calculados e enviados para a saída na próxima borda de subida de clock.
- **reset_in** (1 bit): Torna em zero assíncronamente os dados na saída.
- **clk_in** (1 bit): Entrada de clock.

Saídas:

- **array_data_out** (64 x 16 bits): Saída de dados de 64 posições, cada posição com comprimento de 16 bits.
- **ready_out** (1 bit): Indica que os dados de entrada foram calculados e estão disponíveis na saída.

C. BlockSimpleAdder

Um Bloco com 64 somadores foi criado. Cada somador possui duas entradas de 16 bits e uma saída de 16 bits. Possui também uma entrada de acionamento síncrono (start), um reset assíncrono e uma saída de 16 bits. Isto foi feito para evitar que o bloco infira atrasos de setup e hold. O bloco criado pode ser visualizado na Figura 7. As especificações de entrada e saída são mostradas a seguir.

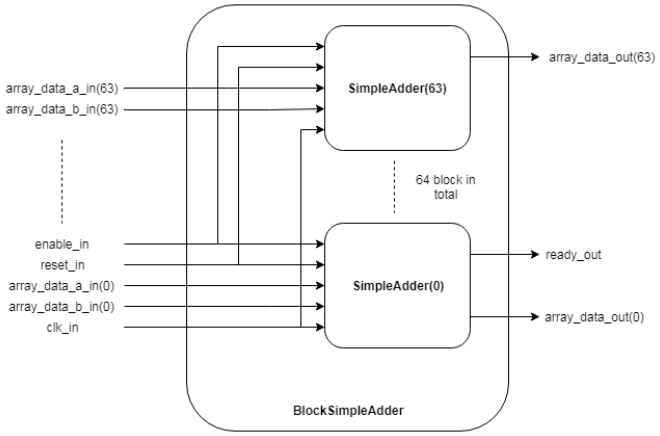

Figura 7. Bloco com 64 somadores de 16 bits em paralelo

Entradas:

- **array_data_a_in** (64 x 16 bits): Primeira entrada de dados com 64 posições de 16 bits cada.
- **array_data_b_in** (64 x 16 bits): Segunda entrada de dados com 64 posições de 16 bits cada.
- **enable_in** (1 bit): Quando enable_in = 1 permite que os dados de entrada sejam calculados e enviados para a saída na próxima borda de subida de clock.
- **reset_in** (1 bit): Quando reset_in = 1, torna zero assíncronamente as saídas do módulo.
- **clk_in** (1 bit): Entrada de clock

Saídas:

- **array_data_out** (64 x 16 bits): Saída de dados com 64 posições de 16 bits cada.
- **ready_out** (1 bit): Quando ready_out = 1, indica que os dados estão disponíveis na saída.

D. Associação dos blocos BlockSignalChanger e BlockSimpleAdder

Um algorítmo que realiza a transformada rápida de Hadamard, também conhecido como **fast Walsh–Hadamard transform** - $FWHT_h$ realiza somas e subtrações de um vetor de tamanho 2^n entregando na saída um vetor com as probabilidades dos termos. Na Figura 8 pode-se visualizar a aplicação da FWHT para um vetor de 8 posições, onde pode-se ver como os somadores devem ter suas entradas ligadas as saídas dos blocos anteriores. Veja que neste caso com o vetor de 8 posições são necessários 3 blocos de somadores para que se obtenha a saída.

Para o caso do problema deste trabalho, o vetor possui 64 símbolos, sendo portanto necessários 6 blocos de somadores, cada bloco com 64 somadores. As entradas e saídas serão associadas de forma semelhante aos da Figura 8. Nenhum somador atuará como subtrator, portanto os dados recebidos nos somadores somente operarão com somas. O bloco que realizará a multiplicação por -1 será o BlockSignalChanger na entrada que realizará a inversão de sinal conforme o valor que será passado para sua entrada **operation_mode_in**.

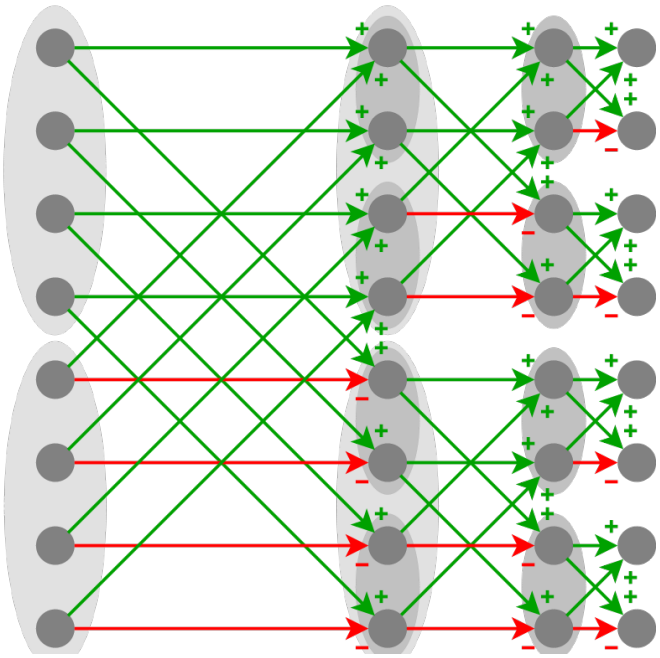

Figura 8. FWHT para um vetor de 8 posições

Para determinar o tamanho que os dados de saída podem atingir, multiplicou-se um vetor de tamanho 64 com cada posição podendo atingir até 2^{10}, por uma matriz de 64 x 64 de 2^{10}, resultando em dados que podem atingir até 65536, ou seja, 2^{16}. Portanto a saída do bloco de somadores poderia atingir, na pior das hipóteses, até 16 bits.

Um bloco chamado de FWHT foi criado para realizar as operações para um vetor de 64 bits. Este bloco pode ser visualizado na Figura 9. Para associar as entradas e saídas de cada módulo dos blocos somadores, foi necessário criar um código no MATLAB, denominado **hadamard_test.m** que entrega a ligação de cada entrada e saída dos seis módulos somadores e da saída do bloco BlockSignalChanger com o as entradas do primeiro somador.

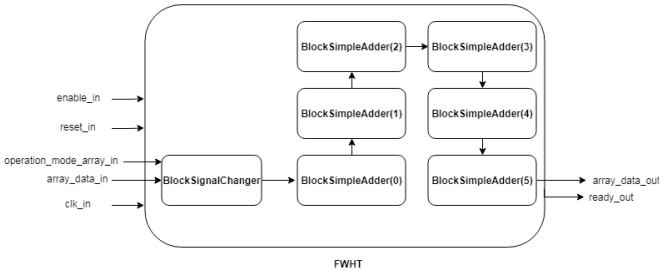

Figura 9. Bloco FWHT para um vetor de 64 posições, cada posição com 16 bits

A entradas e saídas para o bloco FWHT são especificadas a seguir: **Entradas:**

- **enable_in** (1 bit): Quando enable_in = 1, então as entradas serão habilidatas no bloco na próxima borda de subida de clock.
- **reset_in** (1 bit): Quando reset_in = 1, todas as saídas se tornarão zero assíncronamente.
- **operation_mode_array_in** (64 bits): Se operation_mode_array_in = 1 na saída do bloco BlockSignalChanger a saída n será enviada como seu complemento de 2, se operation_mode_array_in = 0 então a entrada n será enviada para a saída.
- **array_data_in** (64 x 16 bits): Entrada de dados de 64 posições, cada posição de 16 bits.

Saídas:

- **array_data_in** (64 x 16 bits): Saída de dados de 64 posições, cada posição de 16 bits.
- **ready_out**(1 bit): Quando ready_out = 1, indica que os dados calculados estão disponíveis na saída.

E. Próximas Etapas da Arquitetura de Hardware

Para o relatório final, espera-se que os blocos FWHT sejam integrados à FIFO de forma a verificar o funcionamento do sistema em conjunto. Serão teoricamente 128 blocos FWHT, onde cada bloco representa um tipo de header que pode ter sido enviado pelo transmissor, e cada header deste determinará um valor diferente para a entrada **operation_mode_in**. Desta forma, não foi possível cumprir este requisito para este ponto de controle, devido a complexidade de se determinar cada header de 64 bits possível de ser codificado.

IV. Resultados

Algumas simulações já foram realizadas para a verificação do funcionamento dos blocos individualmente.

A. Testbench de funcionamento da FIFO

Na Figura 10 pode-se verificar o Testbench da First In First Out. Um contador gerou as entradas a serem enviadas para a FIFO e foi verificado que esta funcionou conforme o

especificado, sempre atualizando os registradores cada vez que foi acionada na borda de subida de clock e também indicando quandos os dados de saída já estavam prontos.

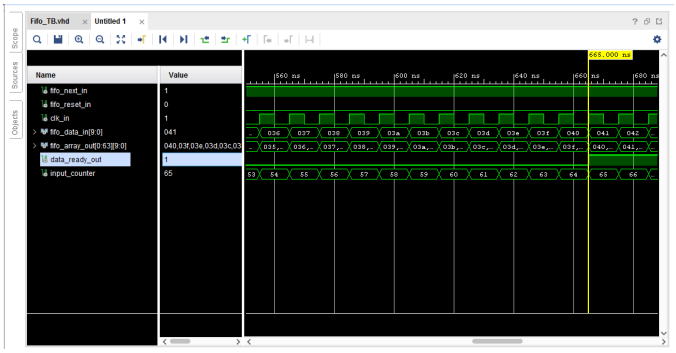

Figura 10. Testbench da FIFO de 64 posições

B. Testbench de funcionamento do bloco SignalChanger

A Figura 11 mostra o Testbench do bloco SignalChanger. O bloco funcionou conforme o esperado e calcula o complemento de 2 da entrada sempre que a entrada operation_mode_in = 1 corretamente. A latência e o throughput são iguais a 1 ciclo de clock.

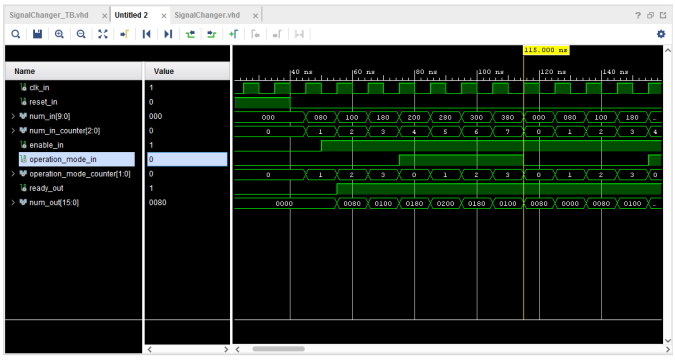

Figura 11. Testbench do bloco Signal Changer

C. Testbench do módulo FWHT com a FIFO

O Testbench deste dois módulos em conjunto está sendo realizado, mas para isto é necessário que se saiba quais são os headers possíveis de serem enviados para se obter o *generate* de cada bloco FWHT. Esta verificação ainda está sendo feita com o estudo da norma DVB-S2X.

V. Conclusões

Conseguiu-se até este momento criar todos os blocos necessários para determinar o header mais provável de ter sido enviado pelo transmissor. É necessário ainda um estudo mais detalhado para verificar quais são os headers possíveis de terem sido enviados e com isto o bloco final que determina o header enviado com base na maior correlação com header de referência pode ser implementado. Apesar de a solução final ser determinada com blocos razoavelmente simples, a grande demora está em entender o problema e organizar as ideias de forma a determinar uma ou mais soluções para o problema. Este inclusive está sendo o maior desafio deste momento, que é determinar os headers de referência com base nas normas DVB-S2 e DVB-S2X. Espera-se que até o relatório final todos os módulos estejam integrados, que a simulação esteja funcionando correntamente e que se consiga gerar um bitstream para teste do módulo final na FPGA Basys 3, de forma a validar a solução construída. Espera-se ainda determinar o consumo de recursos lógicos e de energia da FPGA.

Referências

[1] ROHDESCHWARZ, Tecnologia DVB-S2. Acesso em 11/04/2019. https://www.rohde-schwarz.com/br/tecnologias/transmissao-por-satelite/dvb-s2/tecnologia-dvb-s2/tecnologia-dvb-s2_55598.html

[2] DVB: Uma visão da tecnologia DVB. Acesso em 11/04/2019. http://www.teleco.com.br/tutoriais/tutorialdvb/pagina_3.asp

[3] ETSI EN 302 307-2, v1.1.1, (2014-10), Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 2: DVB-S2 Extensions (DVB-S2x)

[4] Todd K. Moon, Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience. 2005. ISBN: 0471648000.