Relatório Final
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 8 bits através de um gerador de PLS CODE, que pode ser visualizado na Figura 1.

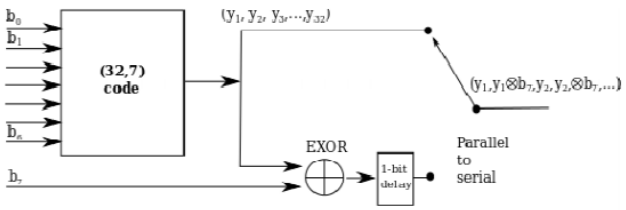

Figura 1. Gerador do PLS CODE

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.

Com os 8 bits são geradas 256 possibilidades de PLS CODE a serem enviados. Dessa forma, seriam necessários 128 cálculos de correlação, gerando um custo computacional enorme. Uma alternativa para este problema é utilizar um que

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

Figura 2. Matriz Reed-Muller

01110001100111011000001111001001010100110100001000101101111110101

Figura 3. Sequência de 64 bits para gerar o embaralhamento

utiliza a ideia Transformada Rápida de Hadamard [4], que fornece um vetor de números que indicam a 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 (definido pelo trabalho), 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 o cálculo de correlação.

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* - com dois blocos de somadores, cada um com 128 posições, onde cada posição possui 10 bits. Um bloco de 128 registradores é utilizado para armazenar os símbolos I e outro bloco de 128 registrados os símbolos Q. A Figura 4 mostra o diagrama de blocos criado. Os 128 registradores de 10 bits são cascateados, de modo que estes são acionados ao mesmo tempo pela entrada **fifo_next_in**. Dos 128 registradores, somente 64 destes são disponibilizados na saída. Portanto, da posição 0 até a 63 os valores não são disponibilizados na saída e de 64 a 127, são disponibilizados 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 **symbol_i_in** e **symbol_q_in** seja enviado para o registrador 0. Também quando **next_in** = '1', os arrays de saída **array_symbol_i_out** e **array_symbol_q_out** são atualizados com os novos valores de símbolos.
- **fifo_reset_in** (1 bit): Quando '1', torna assincronamente todas as saídas dos registradores em 0.
- **clk_in** (1 bit): Entrada de clock do componente.
- **symbol_i_in** (10 bits): Entrada de dados de 10 bits.
- **symbol_q_in** (10 bits): Entrada de dados de 10 bits.

Saídas:

- **data_ready_out** (1 bit): Indica que 64 símbolos de I e Q estão disponíveis para serem lidos.
- **array_symbol_i_out** (64 x 10 bits): Saída de dados que possui 64 valores, cada um com 10 bits.
- **array_symbol_q_out** (64 x 10 bits): Saída de dados que possui 64 valores, cada um com 10 bits.

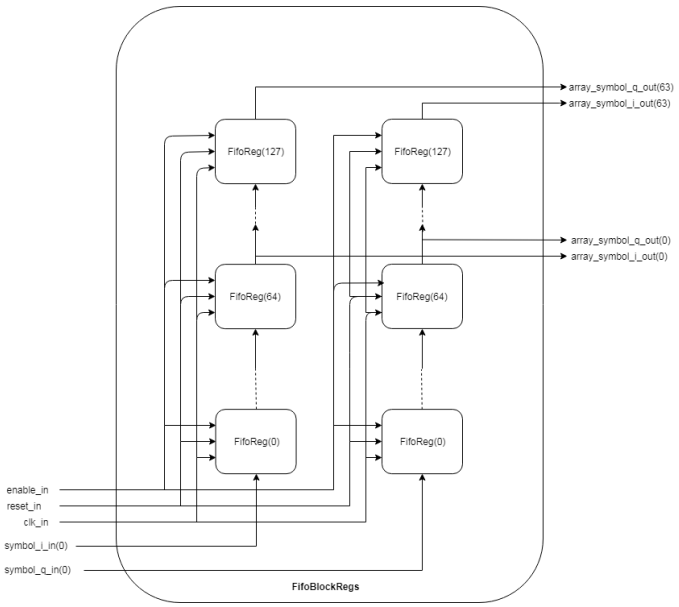

Figura 4. Diagrama de Blocos da *First In First Out*

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 na Seção III-E.

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. 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 de todas as saídas e sinais em 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.

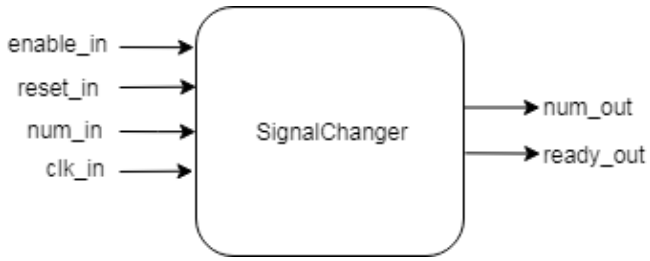

Figura 5. Bloco SignalChanger

C. BlockSignalChanger

Um bloco de 64 SignalChanger, chamado de BlockSignalChanger é 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.

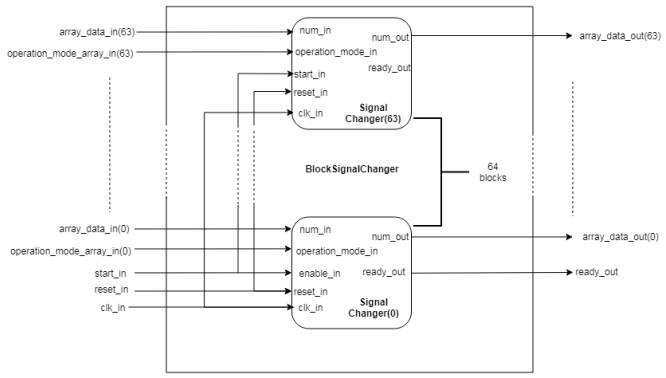

Figura 6. 64 SignalChanger em paralelo, bloco denominado BlockSignalChanger

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 (ou multiplicado por -1) 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.

D. BlockSimpleAdder

Um Bloco com 64 somadores também foi criado. Cada somador possui duas entradas de 16 bits e uma saída de 16 bits. Possui também uma entrada de acionamento síncrono (enable_in), um reset assíncrono (reset_in) 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.

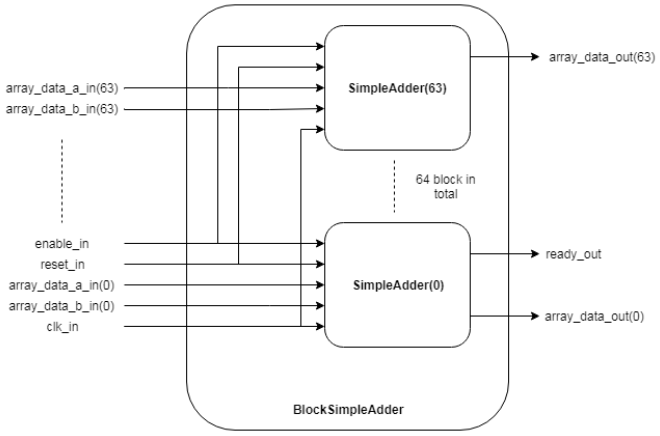

Figura 7. Bloco com 64 somadores de 16 bits em paralelo

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.

E. SignalCorrelator

Para que se possa entender o procedimento de operações deste bloco, deve-se entender primeiramente o algorítmo utilizado. 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 os pequenos blocos somadores de cada módulo devem ter suas entradas ligadas as saídas dos blocos anteriores, com uma associação que varia conforme se caminha pelas ”camadas”de blocos somadores. No caso ainda do exemplo da Figura 8, pode-se verificar que com o vetor de 8 posições são necessários 3 blocos de somadores para que se obtenha a saída. Para obter o número de blocos somadores, portanto deve-se tirar o logarítmo na base 2 do tamanho do vetor de entrada.

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. A única diferença é que nenhum somador atuará como subtrator, portanto os dados recebidos nos somadores somente operarão como somas.

O bloco que realizará a multiplicação por -1 será o **BlockSignalChanger**, mencionando na Seção III-C. Este bloco realizará a inversão de sinal conforme o valor que será passado para sua entrada **operation_mode_in**. Mais especificamente, se o bit n do PLSCODE de 64 bits gerado for ’0’, então o valor recebido na posição n **sempre** será invertido. Se o bit na posição n do PLSCODE for 1, logo não será invertido. Portanto, o PLSCODE n que deseja-se calcular a correlação será passado como argumento para a entrada **operation_mode_in**.

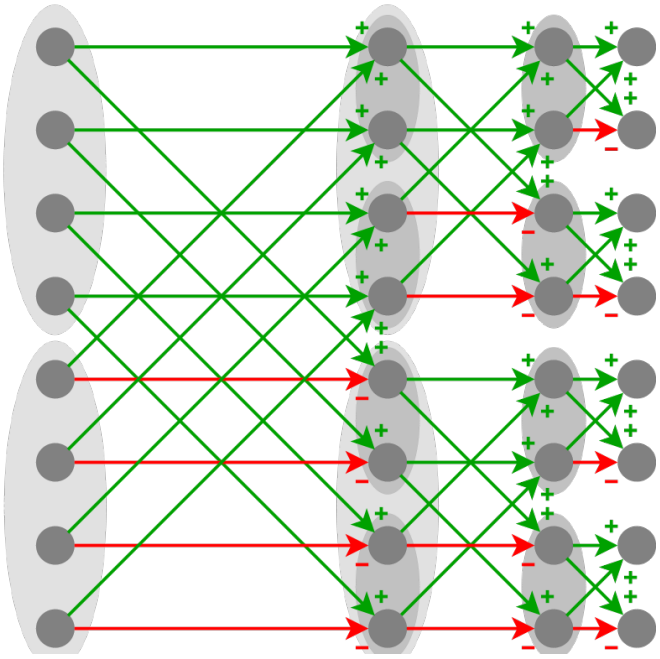

Figura 8. FWHT para um vetor de 8 posições

Neste ponto, faz-se necessário mencionar alguns scripts e funções utilizados no MATLAB para a geração dos blocos que realizam a correlação.

O primeiro script é chamado de *possiblePlsCodes.m*. Este script têm todos os tipos de modulação, taxas de código e tipos de frame suportados pelo decodificador DVB-S2X utilizado no projeto, além do PLSCODE de 7 bits para cada tipo de PLSCODE suportado. Todos esses dados são armazenados em um vetor de structs de forma a serem consultados quando necessário.

O segundo script é uma função chamada de *generate64bitPlsCode.m*. Este script gera a partir do vetor de struct com as informações dos PLSCODES de 7 bits, um PLSCODE codificado em 64 bits conforme determina o procedimento da norma. Após a obtenção de todos os PLSCODES de 64 bits possíveis, estes são armazenados no vetor de structs como um novo campo que pode ser consultado.

O terceiro script é chamado de *CorrelatorGenerator.m*. Este script cria os códigos em VHDL para os blocos *SymbolCorrelator.vhd*, *ComplexModule.vhd* e *Correlator.vhd*.

O quarto script interessante de se mencionar é o *CorrelatorBlockGenerator.m*. Este script gera os arquivos *BlockCorPkg.vhd* que é um arquivo de constantes com todos os PLSCODES de 64 bits possíveis de serem utilizados e o módulo *BlockCorrelators.vhd*.

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 ”uns”no MATLAB, 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 **SignalCorrelator** 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, o script no MATLAB chamado de *CorrelatorGenerator.m* relaciona a ligação de cada entrada e saída dos seis módulos somadores e da saída do bloco BlockSignalChanger com o as entradas do primeiro somador, utilizando a mesma lógica já explicada na Figura 8.

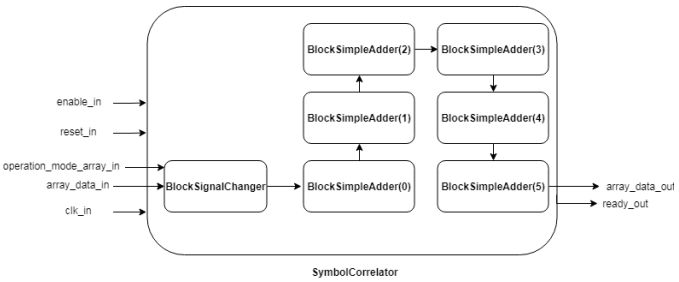

Figura 9. Bloco SymbolCorrelator para um vetor de 64 posições, cada posição com 10 bits

A entradas e saídas para o bloco SymbolCorrelator 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(n)** = 1 na saída do bloco BlockSignalChanger a saída n será enviada como seu complemento de 2, se **operation_mode_array_in(n)** = 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 10 bits.
- **clk_in** (1 bit): Entrada de clock.

Saídas:

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

F. ComplexModule

Este bloco está responsável por calcular o módulo ao quadrado do resultado após o bloco de somas do símbolo I e do símbolo Q. Como as somas realizadas se traduzem na multiplicação do vetor de símbolos por uma matrix de ”1”s de 64 x 64, o vetor de saída no último bloco de somadores possui todos os 64 valores iguais. Desta forma, utiliza-se o array_symbol_out(0) para o símbolo I na primeira entrada de dados do bloco e o array_symbol_out(0) para o símbolo Q na segunda entrada de dados do bloco. Os valores são elevados ao quadrado e somados. Depois disso, são disponibilizados na saída.

As entradas do módulo são:

- **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.
- **clk_in** (1 bit): Entrada de clock.
- **symbol_i_in** (16 bits): Valor resultante de correlação do símbolo I.
- **symbol_q_in** (16 bits): Valor resultante de correlação do símbolo Q

E as saídas são:

- **ready_out** (1 bit): Quando ready_out = 1, o valor de saída em module_out está disponível para ser lido pelo próximo bloco.
- **module_out** (32 bits): Valor do módulo ao quadrado calculado

G. Correlator

O bloco correlator é simplesmente um bloco que associa os blocos **SymbolCorrelator** para os dois vetores de símbolos de entrada com o bloco **ComplexModule**, disponibilizando na saída o módulo que representa a correlação dos símbolos com o PLSCODE escolhido. A Figura 10 mostra como é a ligação dos blocos. As entradas são as seguintes:

- **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.
- **array_symbol_i_in** (64 x 10 bits): Vetor de entrada com os símbolos I.
- **array_symbol_q_in** (16 bits): Vetor de entrada com os símbolos Q.
- **operation_mode_array_in**: Entrada utilizada para receber o PLS CODE de 64 bits invertido (NOT).

E as saídas são:

- **ready_out** (1 bit): Quando **ready_out** = 1, o valor de saída em **module_out** está disponível para ser lido pelo próximo bloco.
- **module_out** (32 bits): Valor do módulo ao quadrado calculado

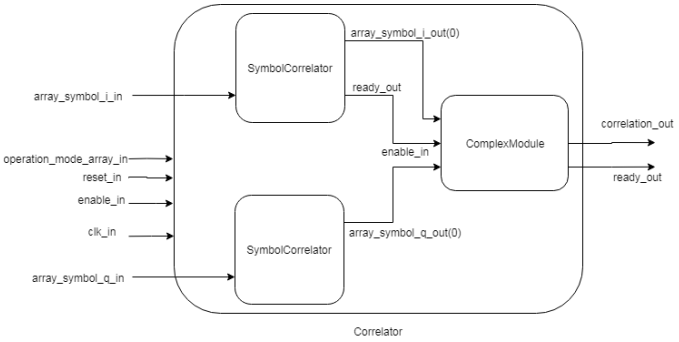

Figura 10. Bloco Correlator: realiza o cálculo de correlação do PLSCODE(n) com os vetores de símbolos disponibilizados nas entradas

H. BlockCorrelators

A Figura 11 mostra a estrutura do Bloco de Correlatores. Este componente tem correlatores para todos os PLHEADERS possíveis de serem enviados. Como já mencionado, os possíveis HEADERS estão no arquivo de constantes **BlockCorPkg.vhd**, criado pelo script no MATLAB *CorrelatorBlockGenerator.m* e são enviados um a um através do parâmetro **generate** para cada um dos correlatores. As saídas dos blocos de correlatores, com os módulos ao quadrado, são associadas no vetor de correlações chamado de **correlation_vector_out**.

As entradas para este bloco são definidas abaixo:

- **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.
- **array_symbol_i_in** (64 x 10 bits): Vetor de entrada com os símbolos I.
- **array_symbol_q_in** (16 bits): Vetor de entrada com os símbolos Q.
- **clk_in** (1 bit): Entrada de clock.

E as saídas são:

- **ready_out** (1 bit): Quando **ready_out** = 1, o valor de saída em **correlation_vector_out** está disponível para ser lido pelo próximo bloco.

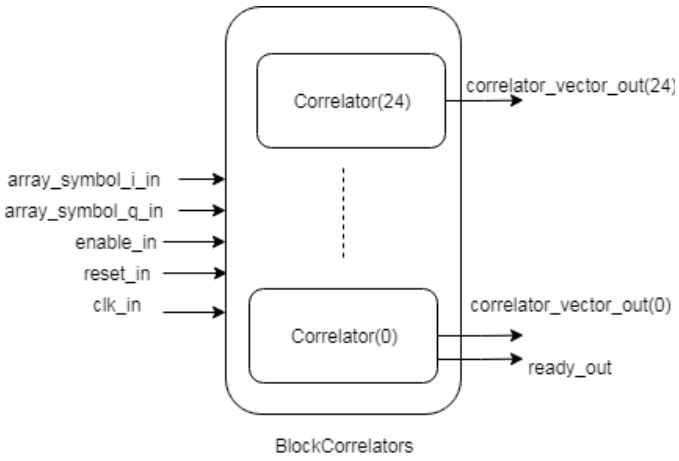

Figura 11. BlockCorrelators: Um bloco que tem correlatores para todos os PLHEADERS possíveis de serem enviados

- **correlation_vector_out** (25 x 32 bits): Vetor com valores de correlação de todos os blocos de correlatores. Por enquanto, foram utilizado 25 tipos de HEADERS diferentes, conforme as informações de tipos modulação e taxas de código utilizados no projeto. **Caso seja necessário aumentar ou diminuir o número de PLHEADERS possíveis de serem enviados ao longo do projeto, os códigos no MATLAB já estão ajustados para esta mudança.**

I. FindHigher

O objetivo deste bloco é percorrer o vetor de correlações geradas e verificar em qual posição está o maior valor. Além disso, deve-se verificar se este valor encontrado supera um valor definido de limiar. Este valor de limiar é definido após testes em software com os valores de correlação gerados para PLSCODES reais enviados e calculados. Definiu-se neste momento que o valor de limiar é 1500. Este valor provavelmente será alterado posteriormente com cálculos mais sofisticados levando em consideração a relação sinal-ruído nos símbolos de entrada. As entradas deste bloco são:

- **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.
- **clk_in** (1 bit): Entrada de clock.
- **correlation_vector_in** (25 x 32 bits): Vetor de entrada com as correlações encontradas.

E as saídas são:

- **ready_out** (1 bit): Quando **ready_out** = 1, indica que foi encontrado uma correlação entre os dados enviados na entrada do módulo e um PLHEADER. A saída higher_value_position_out será enviada para o próximo bloco.
- **higher_value_position_out** (5 bits): Este vetor indica qual correlator encontrou uma correlação com os valores de entrada que superou o limiar definido.
- **no_correlation_found** (1 bit): Indica que foi procurado em todos as posições do vetor um valor de correlação que superasse o limiar e que este valor não foi encontrado. Esta saída serve para alertar os módulos exteriores que uma nova entrada (ou seja, símbolos I e Q na entrada da FIFO) pode ser enviada.

J. Lut

Uma LookUp Table foi implementada para associar a posição de um correlator com um tipo de modulação, taxa de código e tipo de frame. Quando uma correlação que supera o limiar é encontrada, esse bloco é acionado, enviando para a saída os dados do PLHEADER provável de ter sido enviado. Esse bloco também foi criado por um script no MATLAB. Esse script é chamado de *generateLutFile.m* e cria o arquivo **Lut.vhd**.

As entradas deste bloco são:

- **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.
- **clk_in** (1 bit): Entrada de clock.
- **position_in** (5 bits): Vetor de entrada com a posição no vetor de correlações onde foi encontrada a maior correlação.

E as saídas são:

- **ready_out** (1 bit): Quando **ready_out** = 1, as saídas estão disponíveis para serem enviadas para os outros blocos que dependem do PlHeaderDecoder.
- **pilots_out** (1 bit): Tipo que define se os dados enviados possuem *pilots*. Se possuem, logo **pilots_out** = 1. Se não, **pilots_out = 0**.
- **mod_out** (Tipo T_DVBS2X_MOD): Tipo que define que tipo de modulação será enviado.
- **cod_out** (Tipo T_DVBS2X_COD): Tipo que define que taxa de código será enviada.
- **type_out** (Tipo T_DVBS2X_FRAME_TYPE): Tipo que define o tamanho do FRAME que será enviado.

K. DvbS2xPlHeaderDecoder

O bloco final chamado de **DvbS2xPlHeaderDecoder** junta os blocos **Fifo**, **BlockCorrelators**, **FindHigher** e **Lut**.

O bloco recebe varios símbolos I e Q na sua entrada e calcula a correlação com os PLHEADERS possíveis de serem enviados. Enquanto não encontra correlação, vai pedindo que mais símbolos sejam enviados até que um PLHEADER seja encontrado.

As entradas são:

- **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.
- **clk_in** (1 bit): Entrada de clock.
- **symbol_i_in** (10 bits): Entrada de 10 bits do símbolo I.

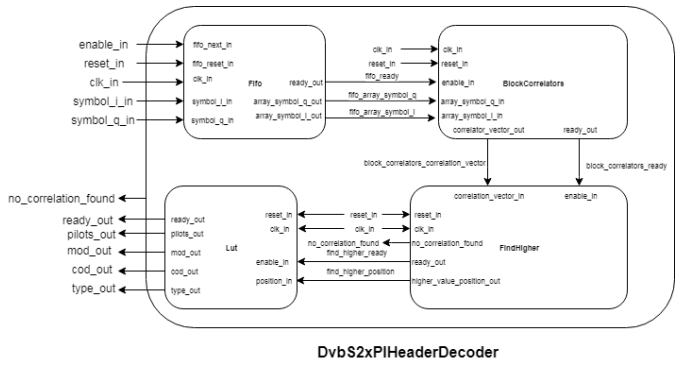

Figura 12. Bloco final DvbS2xPlHeaderDecoder: O bloco recebe os símbolos I e Q e verifica a correlação com os PLHEADERS existentes

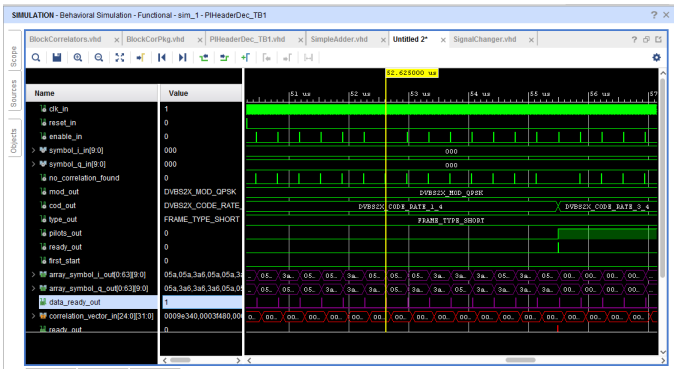

Figura 13. Simulação do bloco final DvbS2xPlHeaderDecoder quando enviado o PLSCODE para a modulação QPSK, taxa de código de 1/4. O decodificador conseguiu detectar após 55 us o PLHEADER enviado

- **symbol_q_in** (10 bits): Entrada de 10 bits do símbolo Q.

As saídas são:

- **ready_out** (1 bit): Quando ready_out = 1, as saídas estão disponíveis para serem enviadas para os outros blocos que dependem do PlHeaderDecoder.
- **pilots_out** (1 bit): Tipo que define se os dados enviados possuem *pilots*. Se possuem, logo pilots_out = 1. Se não, pilots_out = 0.
- **mod_out** (Tipo T_DVBS2X_MOD): Tipo que define que tipo de modulação será enviado.
- **cod_out** (Tipo T_DVBS2X_COD): Tipo que define que taxa de código será enviada.
- **type_out** (Tipo T_DVBS2X_FRAME_TYPE): Tipo que define o tamanho do FRAME que será enviado.
- **no_correlation_found** (1 bit): Indica que foi procurado em todos as posições do vetor um valor de correlação que superasse o limiar e que este valor não foi encontrado. Esta saída serve para alertar os módulos exteriores que uma nova entrada (símbolos) pode ser enviada.

IV. Resultados

A. Simulação para detecção de um PLHEADER

Neste testbench, 90 símbolos do PLHEADER gerado para a modulação QPSK, taxa de código de 1/4 foram gerados. Um script no MATLAB chamado de *firstTestbench.m* foi criado para gerar os símbolos. Além disso, foram utilizados os scripts: *possiblePlsCodes.m* para gerar os PLSCODEs possíveis, a função *generate64bitPlsCode* para gerar os 64 bits do PLSCODE de 8 bits codificado, a função *generateSymbols* para gerar os símbolos para o PLHEADER gerado para a modulação QPSK 1/4, conforme a seção 5.5.2 da Norma para o DVB-S2 e depois cada símbolos foi convertido para ponto fixo de 10 bits, com 2 bits de parte inteira, 7 bits de parte fracionária, e um bit de sinal utilizando a função **dec2binfip.m**. Os símbolos convertidos foram escritos nos arquivos *testbench1_i_symbols.txt* e *testbench1_q_symbols.txt*. Um testbench foi gerado, denominado de *PlHeaderDec_TB1.vhd* para testar as entradas e verificar a detecção. A Figura 13 mostra o HEADER detectado.

B. Simulação para detecção de vários PLHEADERS

Uma outra simulação foi gerada para a detecção de uma sequência de 10 PLHEADERS. Entre cada PLHEADER foi gerado alguns símbolos aleatórios para verificar a capacidade do bloco correlator de não detectar símbolos sem relação. Os símbolos gerados foram enviados para dois IPs de memória ROM, um com os símbolos I e outro com os símbolos Q, e estas memórias foram conectados ao decodificador através de um Top Module. O testbench funcionou corretamente e o decodificador detectou todos os PLHEADERS gerados. O resultado de simulação é mostrado na Figura 14.

Figura 14. Simulação do bloco final DvbS2xPlHeaderDecoder quando vários PLHEADERS foram enviados. Foi possível detectar todos os 10 PLHEADERS em menos de 120 us.

C. Esquemático Preliminar

Um esquemático preliminar foi gerado e pode ser visualizado na Figura 15.

Figura 15. Esquemático preliminar do bloco junto com as memórias ROM

D. Resultados preliminares de síntese

A Figura 16 mostra a utilização percentual de recursos da FPGA Basys 3, onde é possível verificar que a FPGA em questão não possui recursos de harware suficientes para atender as especificações deste módulo. Novas estratégias estão sendo verificadas, talvez na utilização de somente 1 bloco correlator e como consequência haverá um aumento do tempo necessário para analisar os símbolos enviados e encontrar a correlação.

Resource	Utilization	Available	Utilization %
LUT	47639	20800	229.03
LUTRAM	40	9600	0.42
FF	48509	41600	116.61
BRAM	2	50	4.00
DSP	44	90	48.89
IO	7	106	6.60

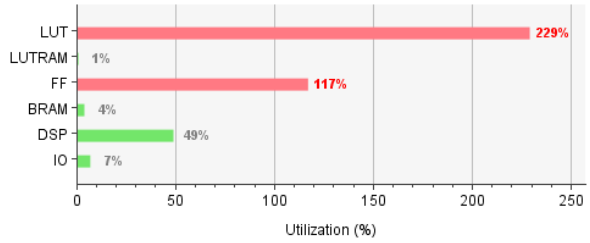

Figura 16. Utilização de recursos da FPGA Basys 3. Pode-se verificar que o bloco não é suportado por este modelo de FPGA.

V. Conclusões

Conseguiu-se até este momento criar todos os blocos necessários para determinar o header mais provável de ter sido enviado pelo transmissor. Simulações preliminares com testbench já foram realizadas, porém ainda é necessário que várias outras simulações sejam realizadas para que o bloco seja validado. Sua implementação na placa e geração de bitstream também são necessários ainda. Apesar disto, já foi possível verificar que os blocos estão cumprindo o que era desejado comparando as saídas obtidas com outros arquivos de simulação do MATLAB. Muitos problemas aconteceram na construção deste bloco, visto que sua complexidade para implementação é altíssima. Foi necessário um estudo bastante detalhado do tópico que aborda o tema na norma, depois várias semanas de implementação e correção dos blocos individuais que compõem o projeto. Além disso, diversos scripts e funções tiveram que ser criados no MATLAB para o teste do bloco. Espera-se que nas próximas semanas o PlHeader Decoder esteja em funcionamento.

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.