MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

IMPLEMENTAÇÃO DE UM DECODIFICADOR SBCDA/ARGOS EM FPGA

Raffael Sadite Cordoville Gomes de Lima(UFRN, Bolsista PIBIC/CNPq)

E-mail: raffael.sadite@crn.inpe.br

José Marcelo Lima Duarte(INPE-CRN, Orientador)

E-mail: jmarcelo@crn.inpe.br

Natal, Rio Grande do Norte.

Julho de 2018

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

IMPLEMENTAÇÃO DE UM DECODIFICADOR SBCDA/ARGOS EM FPGA

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA

(PIBIC/CNPq/INPE)

Raffael Sadite Cordoville Gomes de Lima(UFRN, Bolsista PIBIC/CNPq)

E-mail: raffael.sadite@crn.inpe.br

José Marcelo Lima Duarte(INPE-CRN, Orientador)

E-mail: jmarcelo@crn.inpe.br

Natal, Rio Grande do Norte.

Julho de 2018

Resumo

Este relatório contém a descrição das atividades e estudos desenvolvidos durante o período da bolsa de iniciação científica no INPE/CRN. O objetivo principal deste trabalho foi otimizar os códigos em RTL pré-existentes de um decodificador desenvolvido para o Sistema Brasileiro de Coleta de Dados Ambientais. Apresenta-se neste relatório o que foi modificado e os testes realizados para validação das alterações feitas. A principal contribuição deste trabalho foi a implementação da operação multicanal do demodulador e a posterior integração do mesmo no decodificador. Foi preciso criar um código de verificação em linguagem SystemVerilog para validação do decodificador multicanal possibilitando a verificação completa do hardware. A substituição do bloco de controle automático de ganho por um amplificador variável fez com que uma fosse necessário realizar uma modificação na interface do hardware com o software, tarefa que foi concretizada através do estudo do protocolo de comunicação da interface AHB, utilizada no projeto. Os resultados de síntese de hardware demonstraram que o compromisso de otimização foi alcançado, a diminuição da utilização dos recursos de hardware mostrou que o novo código pode ser sintetizado em um FPGA com menor capacidade o M2S025 da Microsemi, enquanto que o utilizado anteriormente era o M2S050.

SUMÁRIO

LISTA DE FIGURAS

INTRODUÇÃO

Este relatório tem por objetivo descrever as atividades realizadas durante o período compreendido entre setembro de 2017 e julho de 2018. Atividades que compõem o projeto de iniciação científica iniciado em 2013 que consist em no desenvolvimento do equipamento Decodificador DCS para CubeSat, uma carga útil para nanossatélites com a funcionalidade de segmento espacial do Sistema Brasileiro de Coleta de Dados Ambientais (SBCDA) e ARGOS II.

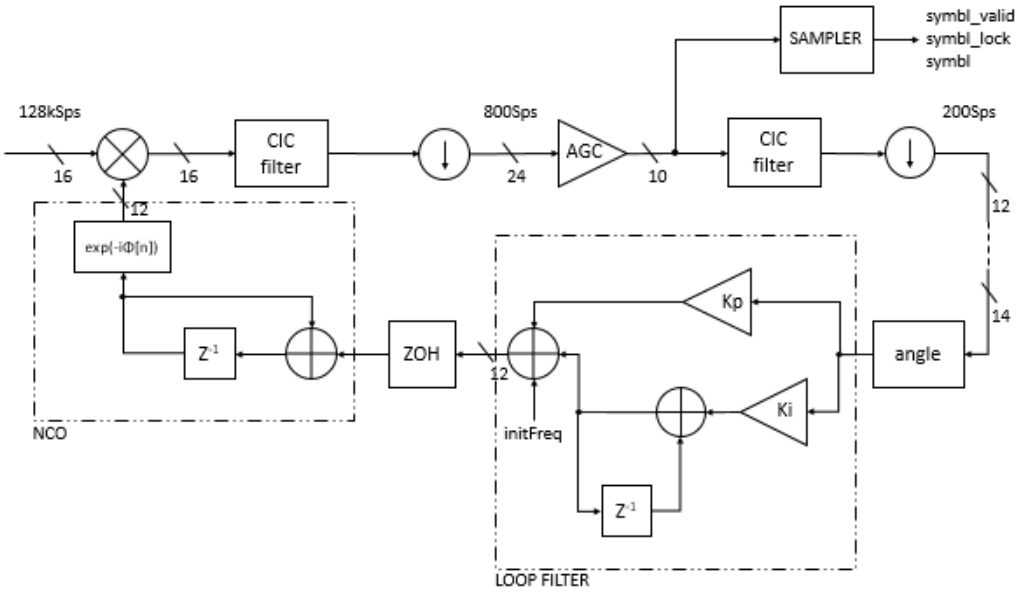
No início deste trabalho, o código RTL do decodificador encontrava-se funcional, os resultados das simulações foram satisfatórios comparados aos do modelo criado em MATLAB. Em um estudo inicial foi possível observar que o modelo poderia ser otimizado, para consumir menos hardware e processar mais sinais em comparação com o código RTL encontrado.

Assim as atividades referentes ao plano de trabalho envolviam o estudo do decodificador implementado em código MatLab e a implementação do demodulador multicanal em RTL, passando pelo estudo da linguagem de descrição de hardware - Verilog, verificação do código implementado através das ferramentas de simulação e síntese ModelSim e LiberoSoC, além da documentação do modelo RTL.

DEMODULADOR

O demodulador é composto por uma malha de recuperação de portadora e uma malha de estimação de tempo para amostragem de símbolo (Litwin, 2001). A primeira ajusta a frequência inicial, dado pelo bloco DETECTOR do decodificador, para a correta demodulação do sinal. A segunda identifica o instante de tempo ótimo para a correta amostragem do símbolo. Podemos observar na Figura 1 o diagrama de blocos do demodulador que havia sido implementado durante o projeto.

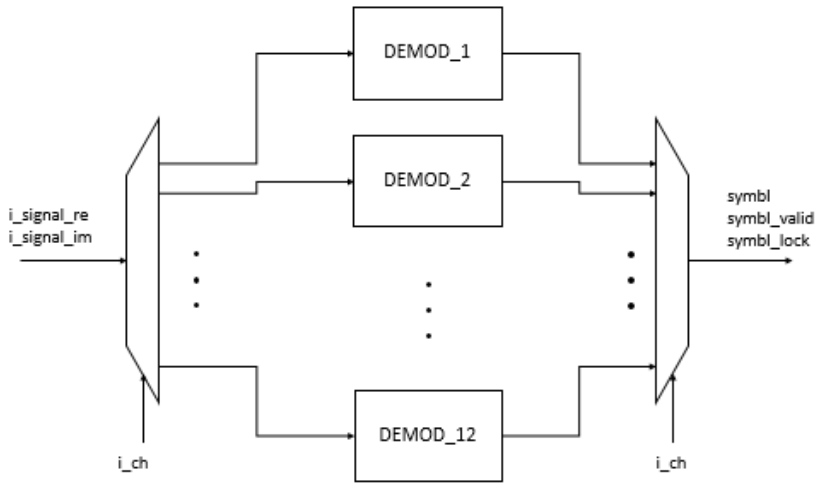
Figura 1: Diagrama de blocos do demodulador inicial.



Fonte 1: Autoria própria.

O demodulador que compõe o decodificador havia sido projetado para o processamento de um sinal por vez, como as Plataformas Terminal Transmissoras (PTTs) não possuem sincronismo na transmissão de dados estima-se que há em média 5 sinais de PTTs coexistindo no tempo o que torna necessário a operação multicanal do decodificador e subsequentemente do demodulador. Por esta razão foi preciso paralelar os demoduladores (como pode ser visto na Figura 2), para que o decodificador fosse capaz de atender a maioria dos sinais que chegam ao decodificador.

Figura 2: Banco de demoduladores.



Fonte 2: Autoria própria.

Desta maneira o hardware necessário para implementação de N canais será equivalente ao hardware necessário para cada canal N vezes.

Sincronismo

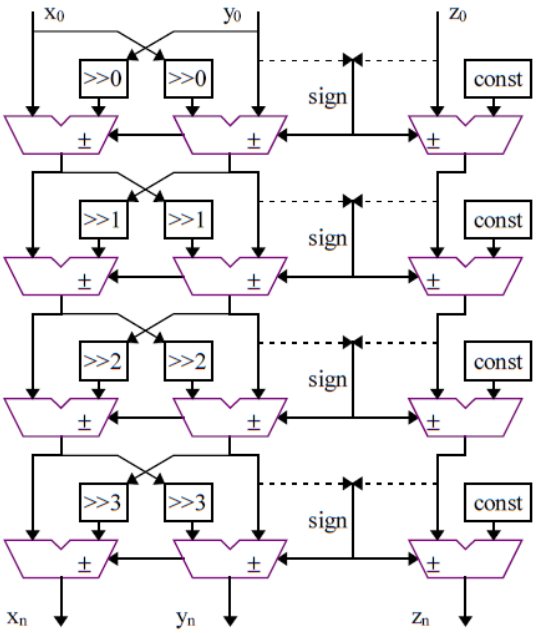
A primeira tarefa, foi realizar o sincronismo entre processamentos das janelas temporais de 10 ms, garantindo que uma JT só comece a ser processada se a anterior tiver sido concluída. Para isto foi necessário implementar uma máquina de estados (FSM) que recebe do SAMPLER o sinal de saída válida e habilita através do sinal *sink_ready* o demodulador a receber novas entradas.

Cordic rotation

Para a aplicação do filtro casado no sinal de entrada é preciso realizar a multiplicação deste sinal pelo sinal da portadora gerada localmente, para uma relação sinal ruído ótima é necessário que a frequência da portadora gerada localmente seja igual à frequência na qual o sinal foi modulado, devido ao enlace espacial, ao efeito Doppler e ao ruído, esta frequência inicial detectada pelo bloco DETECTOR não consegue ser idêntica à original, por isso implementa-se uma malha PLL para ajuste desta frequência. A multiplicação antes feita por um bloco multiplicador complexo, foi modificada e está sendo implementada agora por uma rotação vetorial, através do bloco operacional Cordic (Coordinate Rotation Digital Computer) que utiliza soma/subtração e deslocamentos para tal multiplicação (Andraka, 1998). Foi escolhido a arquitetura pipeline (mostrado na Figura 3) devido ao seu fluxo contínuo

de dados, além disso um banco de registradores foi colocado entre os estágios para quebrar o caminho crítico da operação. A partir de um estudo de convergência pode-se observar que não era necessário mais que oito estágios para o número de bits disponíveis para utilização.

Figura 3: Arquitetura pipeline do Cordic.



Fonte 3: (Andraka, 1998)

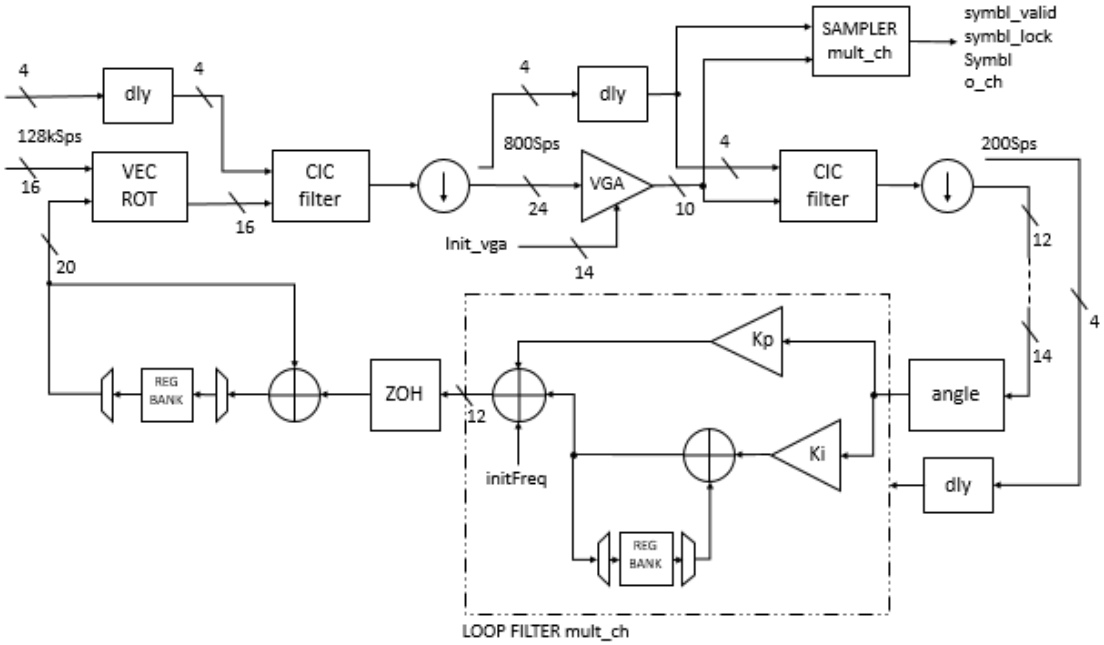
VGA

O bloco VGA (Variable Gain Amplifier) foi utilizado para ajustar o nível de amplitude do sinal após a aplicação do filtro casado, oferecendo um maior aproveitamento da largura de bits disponível para o sinal. Em detrimento com o que havia implementado, este bloco faz o ajuste dado o nível de amplitude do sinal no momento da sua detecção, não há malha de ajuste automático como havia anteriormente. Isto foi modificado para diminuir a complexidade do código RTL e conforme a característica do sinal do ARGOS-2 que possui pequena duração (de no máximo 920 ms).

Multicanal

Por meio do estudo das operações realizadas do demodulador, foi possível obter um modelo que compartilha as operações entre os canais, preservando as operações que necessitam de memória. Dessa forma não sendo necessário a ocupação de hardware em paralelo para atender mais de um sinal ao mesmo tempo. O diagrama de blocos do demodulador desenvolvido neste trabalho pode ser visto na Figura 4.

Figura 4: Diagrama de blocos proposto por este trabalho.



Fonte 4: Autoria própria.

Foi necessário para isto, a utilização de um novo sinal de entrada/saída, o sinal de canal com quatro bits (para implementação de 12 canais) representando qual o canal está sendo processado. Este novo sinal deve sofrer o mesmo atraso que o sinal recebido para correta atribuição na saída da demodulação. As memórias utilizadas e as operações de MUX/DEMUX dependem do sinal de canal.

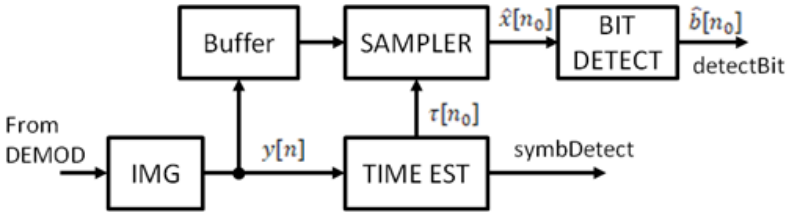
Loopfilter_multch

O filtro de malha é um filtro de segunda ordem responsável pela estabilização da malha de recuperação de portadora (Litwin, 2001), para isto, cada canal deve ter sua malha independente do outro, assim foi preciso um banco de registradores para salvar o erro acumulado de cada canal bem como a parte integrativa do filtro para cada processamento de símbolo e de JT do sinal recebido.

Sampler_multch

O *Sampler* é o bloco responsável pela definição do instante de tempo ótimo para amostragem do símbolo recebido. Podemos visualizar através dos diagramas de blocos das Figuras 5 e 6.

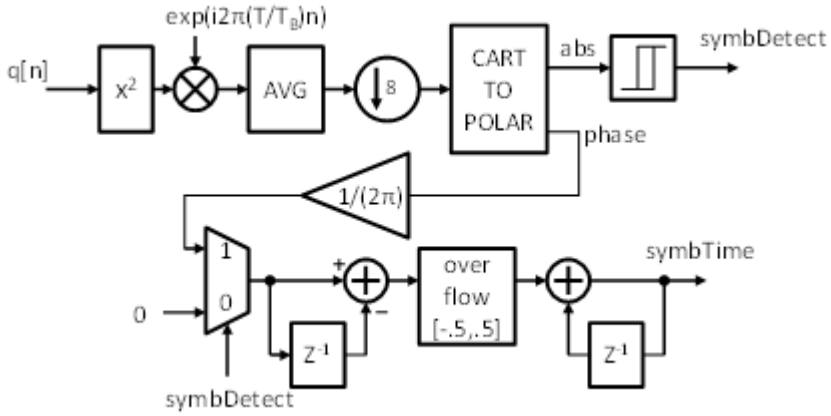
Figura 5: Diagrama de blocos da estimação temporal de amostragem de símbolo.



Fonte 5: Autoria própria.

No bloco de estimação temporal, o *buffer* guarda 4 amostras do sinal advindo da filtragem para escolha do tempo ideal de amostragem. Foi então modificado para que um só bloco de memória fosse utilizado por todos os canais, porém delimitou-se as regiões da memória por canal.

Figura 6: Diagrama de blocos do *Sampler* e Time Est.



Fonte 6: Autoria própria.

No bloco *Sampler* é realizada uma filtragem com decimação (cicfilter_ram_multch), após o deslocamento da frequência do sinal de entrada. Esta filtragem diferentemente das outras neste demodulador, necessita de memória devido ao erro do tempo de amostragem ser cumulativo. Aqui temos um dos maiores blocos de memória do demodulador e assim como anteriormente, foi implementado um único bloco de memória para todos os canais, limitando o acesso a cada canal pelo endereço de memória.

SCRIPT .DO

Para tornar fácil a simulação no ambiente ModelSim foi criado um script em linguagem TCL no formato .do que guarda as informações necessárias para o carregamento de todos os códigos RTL que devem ser compilados, bem como os

dados de entrada e as formas de onda que devem ser visualizadas. Foi feito a partir do capítulo 8 do tutorial fornecido pelo ModelSim (MentorGraphics, 2016), através do caminho: Help>PDF Documentation>Tutorial. Com o script é possível executar a simulação com uma linha de código no *Transcript* do ambiente de simulação. O script está localizado no caminho: C:/Users/Barbara/Desktop/projetos/core/script.

Para isto o usuário deve-se encontrar na pasta de simulação *run*:

```
cd C:/Users/Barbara/Desktop/projetos/core/run
```

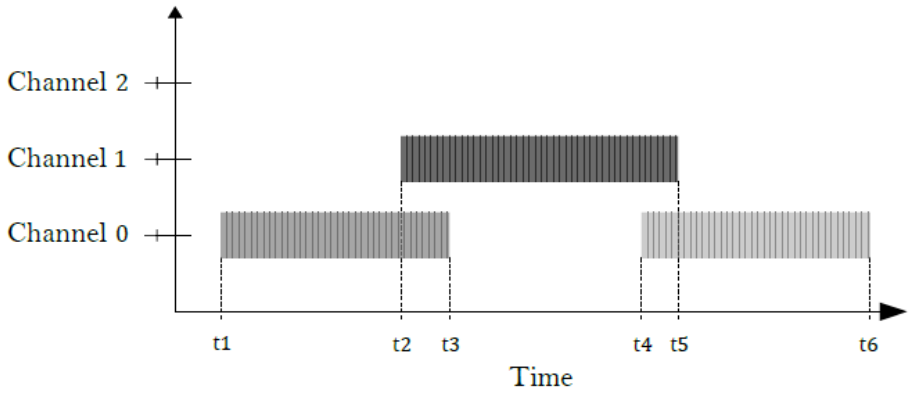
Depois é só executar o script através do comando *source*:

```
source ../script/demod_mult_ch.do
```

SIMULAÇÃO E IMPLEMENTAÇÃO

Para que o sistema fosse testado em condições de igualdade, o código do MatLab foi modificado para realizar a multiplicação por meio das iterações do cordic, e o bloco NCO foi removido, sendo atribuído ao cordic o sinal do acumulo da frequência assim como acontecia no código RTL. Foi realizado um teste padrão com três sinais de PTTs, os quais dois coexistiram no tempo, com amplitudes diferentes e tamanhos iguais (ver Figura 7). Pode-se avaliar a transição entre canais e a transição entre sinais em um mesmo canal.

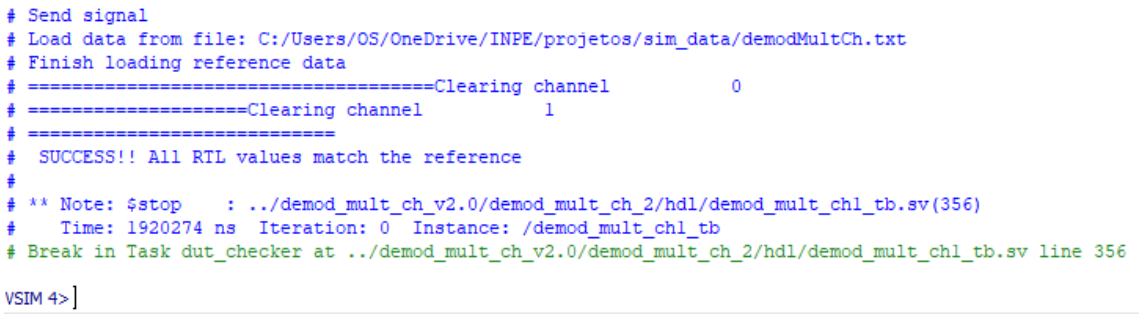
Figura 7: Teste padrão do demodulador multicanal.



Fonte 7: Autoria própria.

O teste comparou os valores obtidos no RTL com os valores de referência calculados pelo MatLab, o resultado pode ser visto na Figura 8.

Figura 8: Resultado da simulação no ModelSim.



Fonte 8: Autoria própria.

A implementação foi realizada no Libero SoC, utilizando o FPGA Microsemi SmartFusion2 M2S025. O resultado mostrou que o código desenvolvido neste trabalho utiliza melhor os recursos de hardware, como pode ser visto na Tabela 1.

Tabela 1: Comparação da utilização dos códigos RTL.

Requisito	Em operação	Proposto	Redução(%)
DSPs	34	18	47%
Total RAM	48	3	93,7%
LUTs	60067	9779	83,7%
SLEs	16807	8863	47,3%

Fonte 9: Autoria própria.

DECODIFICADOR

Após o teste e a implementação, o demodulador multicanal foi integrado ao decodificador. Para isto, foi necessário a modificação do código RTL já criado acrescentando os sinais de canal, que antes era implementado por uma lista da canais ativos, com o acréscimo desta entrada foi necessário modificar a máquina de estados para garantir a operação de *clear* depois que um canal concluísse o processamento de um sinal. A máquina de estados do decodificador possui dois estados, a operação de demodulação e a operação de detecção. O *clear* do demodulador teve que ser efetuado durante a operação de detecção, após a conclusão da demodulação de um sinal, e antes da atribuição de um novo sinal para o canal que estava sendo utilizado.

As entradas de mantissa e expoente utilizados no bloco VGA também tiveram que ser acrescentadas, estes sinais serão atribuídos pelo software do decodificador, portanto não foi preciso nenhuma alteração de lógica para a implementação destes novos sinais.

Testbench

Com o decodificador implementado, foi necessário criar um código de teste em linguagem SystemVerilog. Este arquivo possui três instâncias, o ADC Driver lê os sinais de entrada do arquivo texto criado pelo código do MatLab e carrega os sinais de entrada serialmente. O DUT Driver que instancia os sinais que serão atribuídos pelo software do decodificador e o DUT Checker que compara os valores encontrados com os valores calculados pelo MatLab.

SCRIPT .DO

Da mesma forma como no demodulador multicanal, foi feito um script .do para o decodificador.

SIMULAÇÃO E IMPLEMENTAÇÃO

O teste realizado para o decodificador seguiu o teste padrão proposto no demodulador multicanal. O resultado pode ser visto na Figura 9. Os relatórios de utilização de hardware podem ser vistos no Apêndice.

Figura 9: Resultado da simulação do decodificador multicanal.

```
# Load data from file: C:/Users/Barbara/Desktop/projetos/core/sim_data/demodMultChActiveList.txt
# Finish loading activeList
# Send signal
# Load data from file: C:/Users/Barbara/Desktop/projetos/core/sim_data/demodMultCh.txt
# Finish loading reference data
# Load data from file: C:/Users/Barbara/Desktop/projetos/core/sim_data/demodMultChSignal2.txt
# Finish loading input signal
#
#  SUCCESS!! All RTL values match the reference
#
# ** Note: $stop    : ../rtl/decoder_unit_tb.sv(539)
#    Time: 860447095960 ps  Iteration: 0  Instance: /decoder_unit_tb
# Break in Task dut_checker at ../rtl/decoder_unit_tb.sv line 539

VSIM 3>
```

Fonte 10: Autoria própria.

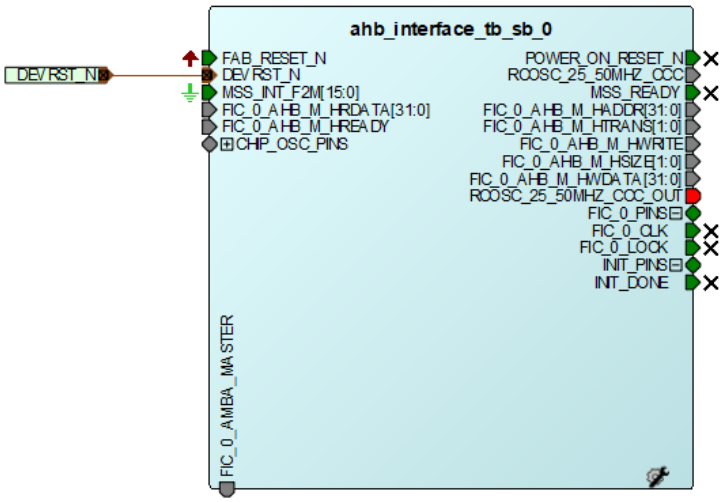
AHB INTERFACE

A comunicação entre software e hardware é feita através da interface AHB (ARM, 2010), como os sinais do bloco VGA são atribuídos pelo microcontrolador, durante a etapa de detecção do sinal transmitido houve a necessidade da modificação desta interface.

Simulação

A simulação da interface AHB foi realizada através do software Libero SoC, por meio da criação de um microcontrolador, na criação do projeto deve-se escolher a opção "Create a microcontroller (MSS) based design" em "Design Template". Clica-se duas vezes no objeto criado, seleciona-se a engrenagem para abrir as configurações de FIC_0, em MSS To FPGA Fabric Interface selecionar AHBLite e User Master Interface. Após salvar, deve-se apertar com o botão direito do cursor em FIC_0_AMBA_MASTER e escolher a opção Show/Hide BIF Pins e selecionar todos os pinos.

Figura 10: Microcontrolador utilizado para realizar a simulação da interface AHB.

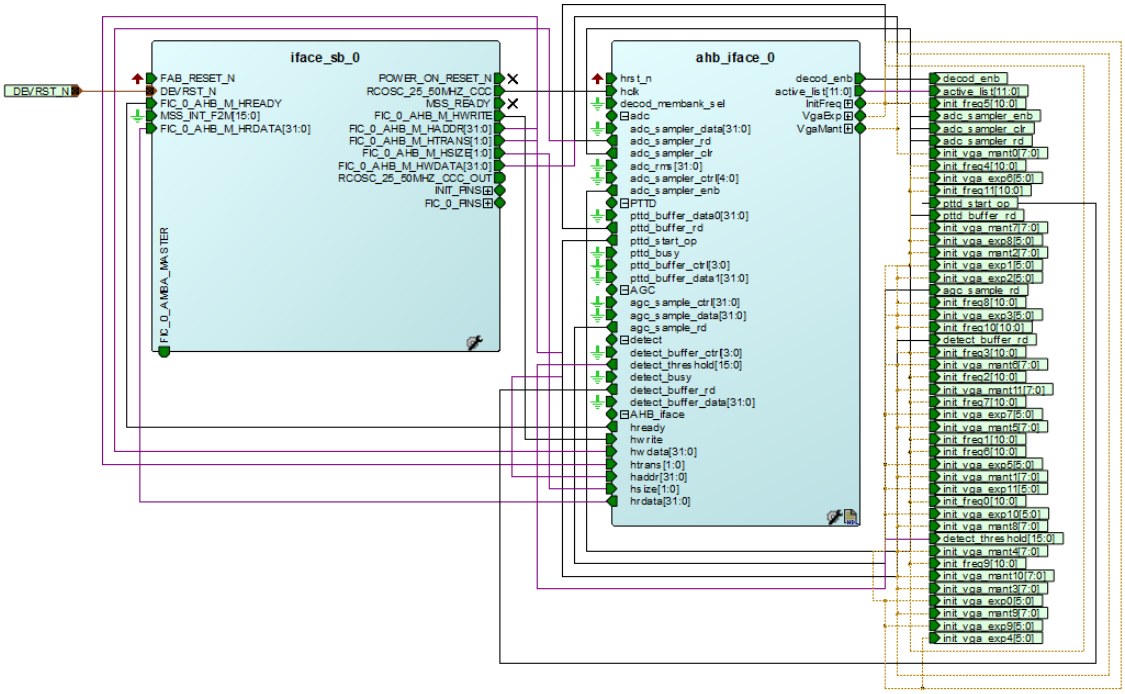


Fonte 11: Autoria própria.

Em File>Import>HDL Source Files adicionar o arquivo ahb_interface.v. Na aba Design Hierarchy clicar com o botão direito no arquivo verilog e selecionar a opção para instanciar o arquivo no Design Flow que possui o nome do projeto que você criou. Deve-se ainda realizar todas as conexões e selecionar o bloco de clock e de

reset na aba Catalog ou configurar o RTC para que este emita o sinal de clock para a interface AHB.

Figura 11: Conexão da interface AHB com o microcontrolador.



Fonte 12: Autoria própria.

Para executar o teste é necessário criar um código no arquivo "user.bfm" que se encontra na aba Files na pasta "simulation" do Libero Soc.

BFM Script

Através do guia do usuário *SmartFusion Bus Functional Model* (Microsemi, 2012) foi possível criar um script simples de escrita e leitura de dados (ver Figura 12), posteriormente foi realizado um script mais complexo, seguindo o fluxo de projeto.

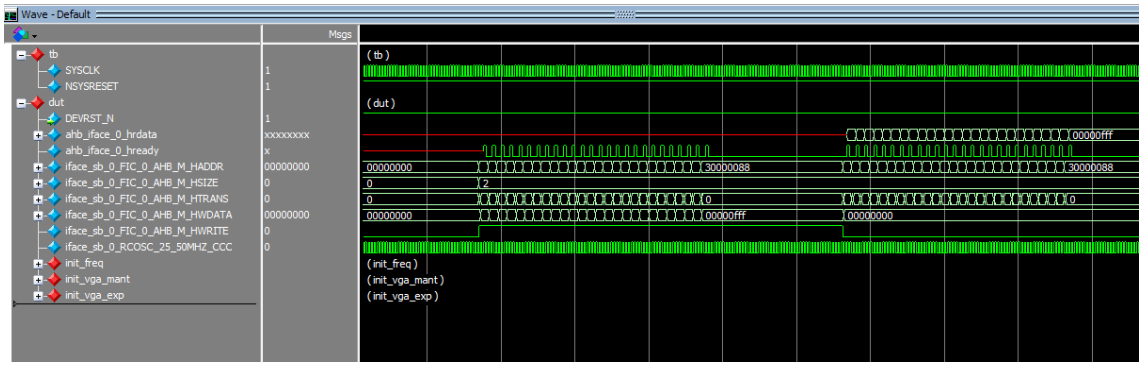
Figura 12: Código executado para simulação da interface AHB.

```
#============================================================
# Enter your BFM commands in this file.
#
# Syntax:
# -------
#
# memmap    resource_name base_address;
#
# write     width resource_name byte_offset data;
# read      width resource_name byte_offset;
# readcheck width resource_name byte_offset data;
#
#============================================================

#include "subsystem.bfm"

memmap  ahbslave 0x30000000;

procedure user_main;
    Header "TEST AHB Interface"

    # perform subsystem initialization routine
      #call subsystem_init;

    # add your BFM commands below:

    #ADC Sampler
    //write w ahbslave 0x04 0x18;
    //write w ahbslave 0x0C 0x11;
    #Active List
    write w ahbslave 0x10 0x07;
    #Init Freq
    write w ahbslave 0x14 0x80;
```

Fonte 13: Autoria própria.

Todos os resultados foram conferidos visualmente através das formas de onda no ModelSim (Figura 13).

Figura 13: Resultado da simulação no ModelSim da interface AHB.



Fonte 14: Autoria própria.

CONCLUSÃO

De acordo com as características dos sinais e das fontes transmissoras foi possível efetuar modificações sensíveis no hardware do demodulador, preservando sua competência.

Todas as modificações propostas permitiram resultados com máxima taxa de acerto quando comparado às saídas de referência obtidas com o código MatLab, a economia de espaço no FPGA resultou na utilização de um modelo menor e mais barato, podendo ser aplicado ao M2S025 da mesma família SmartFusion2, da Microsemi.

Todo o hardware foi integrado, sintetizado e encontra-se na última fase de testes. O software deve sofrer alterações e atualizações para o perfeito casamento do sistema. Os próximos passos do projeto são: a realização dos testes do sistema completo com os sinais de PTTs sendo emitidos pelo gerador de sinais que recebe as formas de ondas do MatLab e a verificação da comunicação com o computador de bordo.

REFERÊNCIAS

Andraka, R. (1998). *A survey of CORDIC algorithms for FPGA based computers*. North Kingstown: Andraka Consulting Group, Inc.

ARM. (2010). *AMBA 3 AHB-Lite Protocol.* www.arm.com.

Litwin, L. (2001). *Matched filtering and timing recovery in digital receivers.* RF design.

MentorGraphics, C. (2016). *ModelSim Tutorial.* Wilsonville: www.mentor.com.

Microsemi, C. H. (2012). *SmartFusion Bus Functional Model - User's Guide.* Aliso Viejo: https://www.microsemi.com/document-portal/doc_view/130908-smartfusion-bus-functional-model-user.

APÊNDICE

Nas figuras 11 e 12 pode-se ver os relatórios de implementação do decodificador com os demoduladores no estágio inicial e após a realização deste trabalho. Foram implementados no FPGA SmartFusion2 M2S050, disponível no laboratório para o número de 12 canais.

Figura 14: Relatório de utilização de hardware do decodificador no estágio inicial deste trabalho.

Resource Usage

Type	Used	Total	Percentage
4LUT	31441	56340	55.81
DFF	13096	56340	23.24
I/O Register	0	801	0.00
User I/O	70	267	26.22
-- Single-ended I/O	64	267	23.97
-- Differential I/O Pairs	3	133	2.26
RAM64x18	39	72	54.17
RAM1K18	12	69	17.39
MACC	72	72	100.00
Chip Globals	3	16	18.75
CCC	1	6	16.67
RCOSC_25_50MHZ	1	1	100.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
MSS	1	1	100.00

Fonte 15: Autoria própria.

Figura 15: Relatório de utilização de hardware do decodificador com o demodulador proposto por este trabalho.

Resource Usage

Type	Used	Total	Percentage
4LUT	17215	56340	30.56
DFF	12827	56340	22.77
I/O Register	0	801	0.00
User I/O	70	267	26.22
-- Single-ended I/O	64	267	23.97
-- Differential I/O Pairs	3	133	2.26
RAM64x18	16	72	22.22
RAM1K18	15	69	21.74
MACC	25	72	34.72
Chip Globals	3	16	18.75
CCC	1	6	16.67
RCOSC_25_50MHZ	1	1	100.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
MSS	1	1	100.00

Fonte 16: Autoria própria.