UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MÁRLON ALLAN LORENCETTI

PARSER EM VHDL PARA DECODIFICADOR DE VÍDEO H.264 PARA SBTVD

Porto Alegre
2010

MÁRLON ALLAN LORENCETTI

PARSER EM VHDL PARA DECODIFICADOR DE VÍDEO H.264 PARA SBTVD

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Engenheiro Eletricista.

ORIENTADOR: Prof. Dr. Altamiro Amadeu Susin

Porto Alegre
2010

MÁRLON ALLAN LORENCETTI

PARSER EM VHDL PARA DECODIFICADOR DE VÍDEO H.264 PARA SBTVD

Este Projeto foi julgado adequado para a obtenção dos créditos da Disciplina Projeto de Diplomação do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Altamiro Amadeu Susin, UFRGS
Doutor pelo Institut National Polytechnique de Grenoble, França

Banca Examinadora:

Prof. Dr. Altamiro Amadeu Susin, UFRGS
Doutor pelo Institut National Polytechnique de Grenoble, França

Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universität Paderborn, Alemanha

Prof. Dr. André Borin Soares, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul, Brasil

Chefe do DELET: _____
Prof. Dr. Marcelo Soares Lubaszewski

Porto Alegre, julho de 2010.

DEDICATÓRIA

Aos meus pais e minha irmã, por toda a dedicação e pelo apoio oferecido para a realização do curso.

AGRADECIMENTOS

Ao professor Susin, pelas oportunidades oferecidas e pela orientação deste trabalho.

À equipe do LaPSI, em especial aos participantes do projeto TV Digital, pela ajuda neste trabalho, pelo convívio e pelo aprendizado durante meus três anos de bolsa.

Ao professor Alberto pela indicação à vaga de bolsista e pela motivação na organização desta disciplina.

À UFRGS e aos professores que direta ou indiretamente participaram da minha formação.

Aos desenvolvedores dos projetos de código aberto usados neste trabalho, dentre os quais se destacam: GHDL, GTKWave, LaTeX, DEleTeX, Dia.

RESUMO

Este documento apresenta o desenvolvimento de uma arquitetura do módulo de parser para um decodificador de vídeo H.264/AVC para o SBTVD. Este módulo é responsável por identificar os elementos do bitstream de entrada, entregando-os aos módulos encarregados de interpretá-los. A arquitetura proposta é capaz de operar em tempo real integrada ao sistema e faz a interpretação dos parâmetros que regem as funcionalidades exigidas pelas normas da ABNT para a decodificação do sinal fonte de vídeo no perfil Baseline, além de oferecer suporte a algumas funcionalidades dos perfis Main e High. A arquitetura é composta por uma interface de entrada para receber os dados de vídeo vindos do demultiplexador no terminal de acesso, módulos de controle, filas do tipo FIFO, decodificador de entropia e os decodificadores de parâmetros de sequência, parâmetros de imagem, cabeçalho de slice e dados de slice. A implementação foi sintetizada para os FPGAs Xilinx Virtex-II Pro XVC2VP30 e Virtex-5 XC5VLX110T.

Palavras-chave: H.264, parser, VHDL, FPGA, SBTVD.

ABSTRACT

This document presents the development of an architecture for the parser module of an H.264/AVC video decoder for the Brazilian Digital Television System (SBTVD). This module is responsible for the identification of elements in the input bitstream, sending the element to the appropriate module for its interpretation. The architecture is able to operate in real time when integrated to the decoder and interprets the parameters that rule the features required for the Baseline profile, and supports some features of Main and High profiles. It consists of a frontend that receives the input video data from the demuxer in the access terminal, control modules, FIFO buffers, entropy decoder and the decoders for sequence parameters, picture parameters, slice header and slice data. The implementation was synthesized for Xilinx Virtex-II Pro XVC2VP30 and Xilinx Virtex-5 XC5VLX110T FPGAs.

Keywords: H.264, parser, VHDL, FPGA, SBTVD.

LISTA DE ILUSTRAÇÕES

Figura 1: Divisão espectral do canal em segmentos 14
Figura 2: Diagrama de blocos do decodificador de vídeo H.264 15
Figura 3: Modos de predição intra 4x4 [1] 17
Figura 4: Modos de predição intra 16x16 [1] 17
Figura 5: Divisão de macrobloco em partições e seus respectivos índices 18
Figura 6: Divisão de partição de macrobloco em subpartições e seus respectivos índices 18
Figura 7: Interface gráfica do PRH.264 31
Figura 8: Placa de prototipação Digilent XUPV2P 32
Figura 9: Placa de prototipação Digilent XUPV5 33
Figura 10: Estrutura geral da implementação do módulo parser 34
Figura 11: Sinais de temporização, de consumo de bits da entrada, de estado atual e de próximo estado 35
Figura 12: Interfaces do módulo desenvolvido para o decodificador de SPS . . . 36
Figura 13: Diagrama de estados do decodificador de SPS 38
Figura 14: Interfaces do módulo desenvolvido para o decodificador de PPS . . . 39
Figura 15: Diagrama de estados do decodificador de PPS 39
Figura 16: Interfaces do módulo desenvolvido para o decodificador de slice header 40
Figura 17: Diagrama de estados do decodificador de slice header 41
Figura 18: Interfaces do módulo desenvolvido para o decodificador de slice data 43
Figura 19: Diagrama de estados do decodificador de slice data 44
Figura 20: Solicitação de coeficientes ao CAVLD 46

LISTA DE TABELAS

Tabela 1: Faixas de *codeNum* 25
Tabela 2: Resultados de síntese do parser completo 48
Tabela 3: Resultados de síntese do decodificador de SPS 48
Tabela 4: Resultados de síntese do decodificador de PPS 49
Tabela 5: Resultados de síntese do decodificador de slice header 49
Tabela 6: Resultados de síntese do decodificador de slice data 49
Tabela 7: Resultados de síntese do decodificador de entropia CAVLD 49
Tabela 8: Resultados de síntese do barrel-shifter 49
Tabela 9: Resultados de síntese do decodificador de Exp-Golomb 49

LISTA DE ABREVIATURAS

ABNT	Associação Brasileira de Normas Técnicas
ASIC	Application Specific Integrated Circuit
CABAC	Context-based Adaptive Binary Arithmetic Coding
CABAD	Context-based Adaptive Binary Arithmetic Decoder
CAVLC	Context-based Adaptive Variable Length Coding
CAVLD	Context-based Adaptive Variable Length Decoder
DVB-SPI	Digital Video Broadcast – Synchronous Parallel Interface
DVD	Digital Video Disc
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
HE-AAC	High-Efficiency Advanced Audio Coding
ISDB-T	Integrated Services Digital Broadcasting - Terrestrial
ISE	Integrated Software Environment
LaPSI	Laboratório de Processamento de Sinais e Imagens
MBAFF	Macroblock Adaptive Frame-Field Coding
MPEG	Moving Picture Experts Group
NAL	Network Abstraction Layer
SBTVD	Sistema Brasileiro de Televisão Digital
SDL	Simple DirectMedia Layer
UFRGS	Universidade Federal do Rio Grande do Sul
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VUI	Video Usability Information

SUMÁRIO

1 INTRODUÇÃO 12

2 CONTEXTUALIZAÇÃO 13
2.1 Sistema Brasileiro de Televisão Digital – SBTVD 13
2.2 Rede H.264 Brasil 14
2.3 Padrão de codificação H.264 15
2.3.1 Macroblocos 15
2.3.2 Parser 16
2.3.3 Predição intra-quadro 16
2.3.4 Predição inter-quadros 17
2.3.5 Decodificadores de entropia 18
2.3.6 Mudança de escala e transformada inversas 19
2.3.7 Filtro de deblocagem 19
2.3.8 Perfis e níveis 19

3 MÓDULO PARSER NO DECODIFICADOR H.264 21
3.1 Unidades de NAL 22
3.1.1 SPS 22
3.1.2 PPS 23
3.1.3 Slices 23
3.2 Codificação Exp-Golomb 24
3.3 Codificação de entropia CAVLC 26
3.4 Codificação de entropia CABAC 26

4 METODOLOGIA E FERRAMENTAS UTILIZADAS 28
4.1 Normas 29
4.1.1 ITU-T 29
4.1.2 ABNT 29
4.2 Simulação VHDL 29
4.3 Geração de estímulos 30
4.3.1 Codificador JM 30
4.3.2 Captura de bitstream das emissoras 30
4.3.3 Geração de resultados esperados 30
4.4 Síntese VHDL 32
4.5 Placa de prototipação 32

5 ARQUITETURA PROPOSTA 34
5.1 Estrutura geral das máquinas de estado 35
5.2 Decodificador de SPS 35
5.3 Decodificador de PPS 37
5.4 Decodificador de slice header 37
5.5 Decodificador de slice data 42
5.6 Módulos herdados 45
5.6.1 Controle do parser 45
5.6.2 Barrel-shifter 45
5.6.3 Decodificador Exp-Golomb 45
5.6.4 Decodificador de entropia CAVLC 45

6 RESULTADOS 47
6.1 Simulações 47
6.2 Síntese para FPGA 47

7 CONCLUSÕES 50

REFERÊNCIAS 52

1 INTRODUÇÃO

Este documento apresenta a arquitetura do módulo de parser utilizada no desenvolvimento de um decodificador de vídeo H.264 para ser usado no SBTVD. A arquitetura proposta foi descrita em linguagem VHDL e sintetizada para os FPGAs Xilinx Virtex-II Pro XVC2VP30 e Xilinx Virtex-5 XC5VLX110T, usados para prototipar o decodificador de vídeo neste projeto.

O capítulo 2 apresenta o contexto de projeto e desenvolvimento do decodificador de vídeo, apresentando a rede nacional de desenvolvimento de tecnologia em TV digital e mostra as características e funcionalidades do padrão H.264.

O capítulo 3 mostra o módulo parser no decodificador, detalhando seu funcionamento e estruturas necessárias à decodificação de vídeo.

O capítulo 4 apresenta as ferramentas e normas utilizadas durante o projeto e verificação do módulo parser.

No capítulo 5 é apresentada a arquitetura de hardware proposta, detalhando sua estrutura.

No capítulo 6 são apresentados as simulações e os resultados obtidos.

As conclusões são apresentadas no capítulo 7.

2 CONTEXTUALIZAÇÃO

Este trabalho faz parte do projeto de um decodificador de vídeo H.264 para o SBTVD. Para compreensão do contexto de desenvolvimento, é feita uma breve introdução sobre o SBTVD, a rede de desenvolvimento de tecnologia das universidades brasileiras e sobre o padrão de codificação de vídeo usado neste sistema.

2.1 Sistema Brasileiro de Televisão Digital – SBTVD

O Brasil passa agora por uma transformação em seu sistema de televisão. O sinal analógico, que começou a ser transmitido no país em 1950 dá lugar a uma nova mídia, a TV digital. Nova mídia pois tem a intenção de mudar a maneira como o brasileiro se relaciona com a TV, deixando de ser um simples espectador e passando a ter participação ativa na programação.

O Sistema Brasileiro de Televisão Digital, ou SBTVD [2], foi criado em 2006. Este sistema, baseado no padrão japonês ISDB-T, traz ao país o título de melhor sistema de TV digital do mundo. Além da interatividade proposta, este sistema usa métodos avançados de codificação de áudio de vídeo, permitindo a recepção do sinal em dispositivos móveis e garantindo uma melhor qualidade de áudio e imagem quando comparado a outros sistemas.

O padrão ISDB-T foi escolhido como base para o SBTVD em comparação feita com os padrões ATSC (americano) e DVB (europeu). Entre as razões para a escolha, destacam-se [2]:

- Modulação COFDM-BST (Coded Orthogonal Frequency Division Multiplexing - Band Segmented Transmission). Este sistema de modulação faz a divisão do canal em treze segmentos, como mostrado na figura 1. O segmento central é usado

para transmitir a programação para celulares e equipamentos portáteis, chamada programação *one-seg*. Diferentemente de outros sistemas de TV digital, no ISDB-T a recepção em dispositivos portáteis é gratuita. Os outros doze segmentos são usados para a programação *full-seg* de alta definição, destinada aos *set-top boxes*, receptores conectados aos televisores.

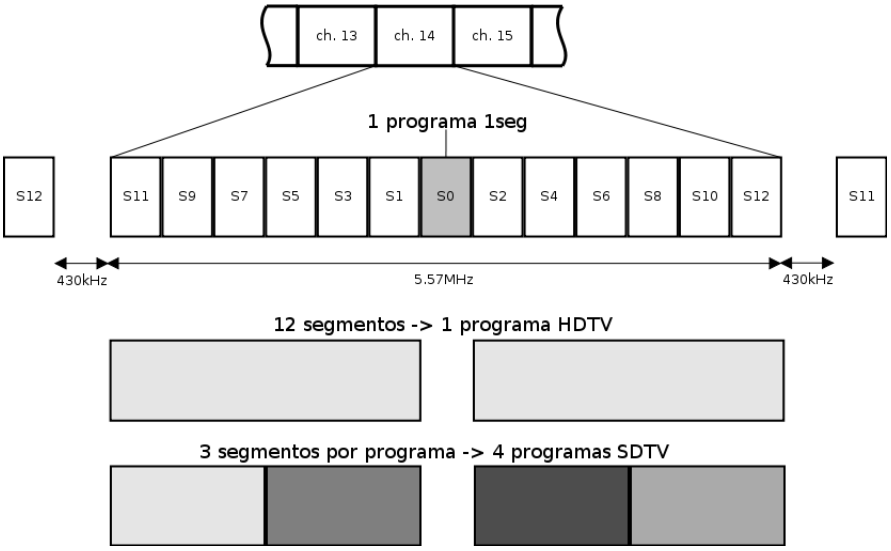

Figura 1: Divisão espectral do canal em segmentos

- Possibilidade de incorporar novas tecnologias. No Brasil a codificação de vídeo é feita usando o padrão H.264 ao invés do padrão MPEG2 usado pelos outros sistemas. Para o áudio usamos HE-AAC que permite a mesma qualidade dos outros sistemas com menor banda de frequência.

- Possibilidade de usar um *middleware* (software de interatividade) nacional, desenvolvido pelas instituições acadêmicas brasileiras.

2.2 Rede H.264 Brasil

A fim de dominar a tecnologia envolvida no SBTVD, o Governo Federal financia, através de órgãos como a FINEP, o desenvolvimento de projetos de pesquisa em várias universidades do país. Todos os processos envolvidos – a codificação dos sinais, a transmissão, a recepção, a decodificação e a apresentação dos mesmos – são cobertos por diferentes grupos de pesquisa espalhados pelo país.

Uma das redes de projeto é a rede H.264 Brasil [3], responsável por implementar dispositivos para a codificação e decodificação de vídeo e áudio do SBTVD. Fazem parte

desta rede as seguintes universidades: UFRGS, UFSC, Unicamp, USP, UFRN, IME, UFRJ, UnB. O CEITEC, fábrica de semicondutores instalada em Porto Alegre, também faz parte desta rede.

O Departamento de Engenharia Elétrica da UFRGS participa desta rede através do Laboratório de Processamento de Sinais e Imagens, o LaPSI, desenvolvendo o decodificador de vídeo em hardware.

2.3 Padrão de codificação H.264

Na codificação de vídeo do SBTVD, o padrão H.264/AVC é utilizado. Tal padrão foi desenvolvido em conjunto pelo grupo MPEG e a ITU-T [4] em 2001. Se comparado ao estabelecido padrão MPEG2 – usado em outros sistemas de TV digital e nos DVDs – apresenta metade da taxa de bits para a mesma qualidade de vídeo. Entretanto, a complexidade computacional do H.264 é consideravelmente maior. Pelo diagrama de blocos do decodificador, mostrado na figura 2, podemos perceber que existe uma grande modularidade das operações, de modo que um hardware específico para efetuar a decodificação torna-se uma alternativa viável para obter desempenho suficiente para realizar a tarefa em tempo real.

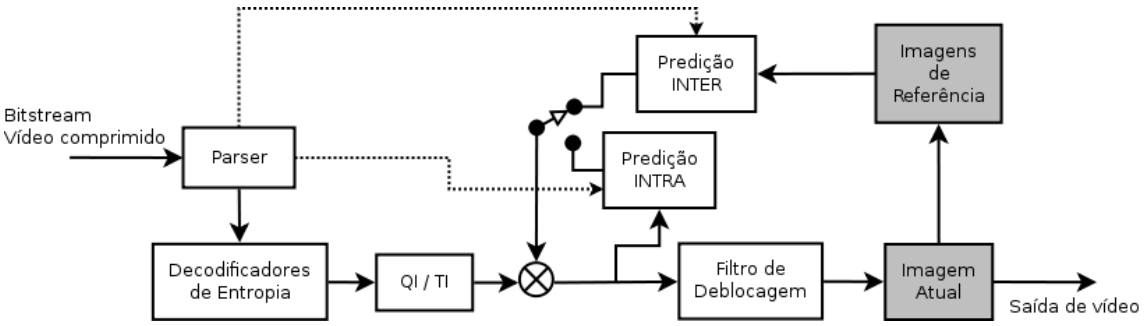

Figura 2: Diagrama de blocos do decodificador de vídeo H.264

2.3.1 Macroblocos

Todo o processamento da imagem é feito em unidades fundamentais chamadas macroblocos, que são blocos de tamanho 16x16 pixels. O espaço de cores utilizado para representação dos pixels é o *YCbCr*. Neste espaço de cores, cada pixel tem uma componente de luminância (Y) e duas de crominância (Cb para azul e Cr para vermelho).

Tendo em vista que o olho humano é mais suscetível a variações de intensidade lu-

minosa que variações de cor, as componentes Cb e Cr podem ser sub-amostradas para reduzir a representação da imagem. A sub-amostragem de cor usada na codificação de vídeo do SBTVD [5] segue o formato 4:2:0, onde para cada quatro amostras de luminância, existe apenas uma amostra para cada componente de crominância.

Para a geração da imagem dos macroblocos, inicialmente é feita uma predição que pode ser tanto baseada em pixels vizinhos do mesmo quadro quanto em pixels de quadros distintos no tempo, do passado ou do futuro. Este processamento é feito, respectivamente, nos módulos de predição intra-quadro e inter-quadros. O uso de predição deve-se à correlação existente nos dados de imagem, tanto espacial quanto temporal. Ao resultado da predição, é somado um resíduo, que consiste na diferença entre a imagem final e a imagem predita.

Reduzidas as redundâncias presentes nos dados de imagem subtraindo o resultado da predição, a compactação é mais eficiente, pois os resíduos são mais descorrelacionados entre si. A fim de reduzir ainda mais a correlação dos resíduos, estes passam por uma transformada espacial. Os coeficientes resultantes são então codificados usando técnicas de codificação de entropia, o que acaba por compactar bastante a representação destes coeficientes.

No decodificador, os resíduos são gerados pelo caminho de dados que passa pelos módulos de parser, decodificador de entropia, quantização inversa e transformada inversa.

2.3.2 Parser

O bitstream de vídeo recebido na entrada tem seus pacotes interpretados pelo módulo do parser, que identifica parâmetros de configuração do decodificador e propriedades do vídeo que está sendo decodificado. As informações desempacotadas gerenciam os diversos módulos do decodificador. O parser, atuando como módulo de controle de fluxo de dados de entrada, identifica elementos sintáticos do bitstream de entrada e então ativa o módulo responsável por utilizar tal elemento. O capítulo 3 explica em mais detalhes o funcionamento deste módulo.

2.3.3 Predição intra-quadro

A predição intra-quadro opera em blocos de 4x4 ou 16x16 pixels e é uma funcionalidade nova em codificação de vídeo, introduzida pelo H.264. A predição é gerada a partir de uma cópia, média ou interpolação dos pixels vizinhos ao bloco atual. Os modos de

predição existentes são mostrados nas figuras 3 e 4, respectivamente para blocos de 4x4 e 16x16 pixels.

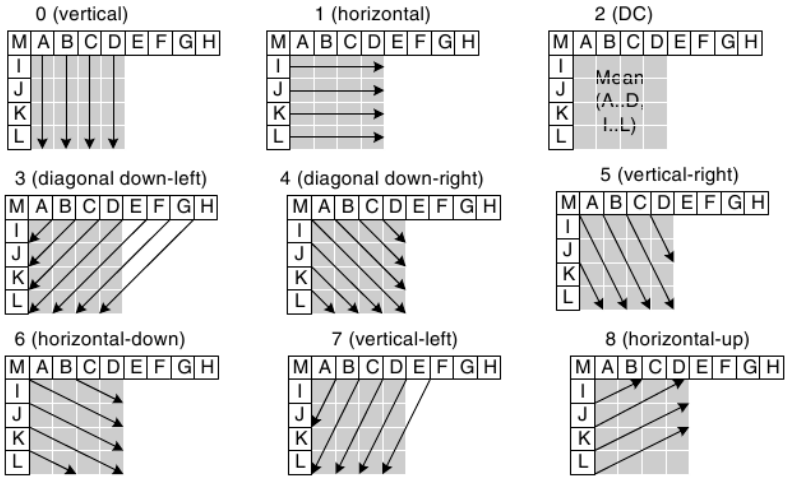

Figura 3: Modos de predição intra 4x4 [1]

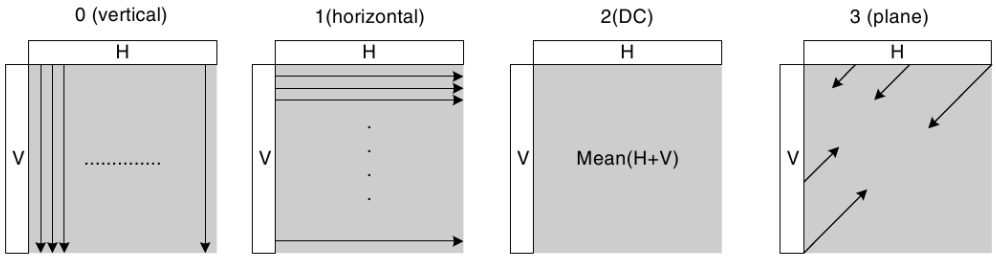

Figura 4: Modos de predição intra 16x16 [1]

2.3.4 Predição inter-quadros

A predição inter-quadros do H.264 é feita a partir de cópias de partes de quadros do passado e/ou do futuro na ordem de exibição, interpoladas com precisão de até um quarto de pixel. Para gerenciar os quadros de referência existem duas listas de referência, onde a *lista 0* guarda os quadros do passado e a *lista 1* os do futuro.

Para a utilização de quadros futuros, a ordem de codificação deve ser alterada para que tais quadros estejam disponíveis para referência no momento da decodificação do quadro atual. Operações de gerenciamento de lista podem alterar a ordem dos quadros, descartá-los ou guardá-los para referências de longo prazo.

Quando a predição é feita com um ou mais quadros do passado, dizemos que o bloco atual é do tipo P. Quando é feita uma bi-predição, ou seja, com um quadro do passado e um do futuro, o bloco é do tipo B. A predição inter-quadros necessita de uma (para tipo

P) ou duas (para tipo B) imagens de referência, que tem suas amostras copiadas da região indicada por vetores de movimento.

Cada macrobloco pode ser dividido em partições, como indicado na figura 5. Para cada partição podem ser usadas imagens diferentes das listas de referência e vetores de movimento distintos, conforme indicação dos elementos sintáticos enviados no bitstream.

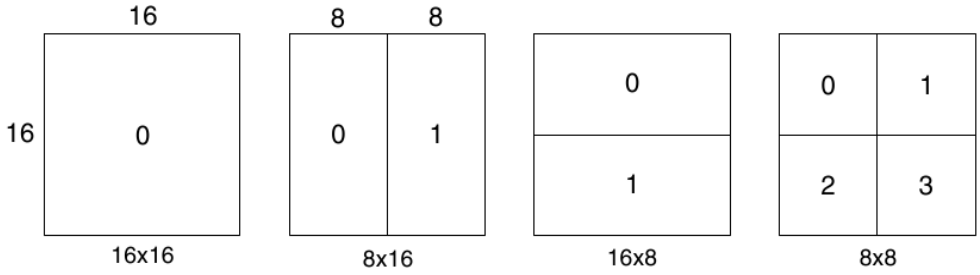

Figura 5: Divisão de macrobloco em partições e seus respectivos índices

Quando o macrobloco for dividido em 4 partições de tamanho 8x8, cada partição pode ser redividida em subpartições, como indicado na figura 6. Cada subpartição pode usar diferentes vetores de movimento, apesar de compartilharem as mesmas imagens de referência indicadas pela partição de macrobloco.

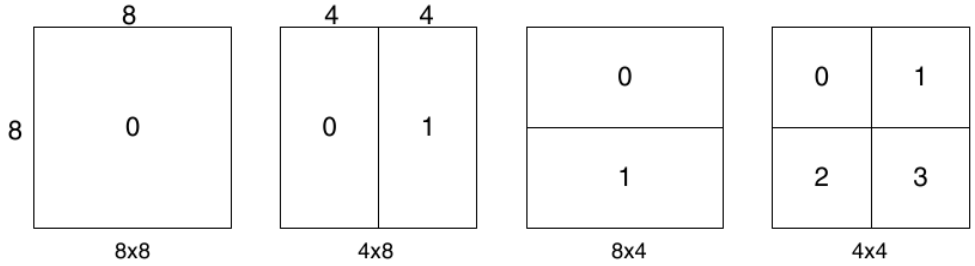

Figura 6: Divisão de partição de macrobloco em subpartições e seus respectivos índices

2.3.5 Decodificadores de entropia

A fim de gerar os resíduos, parte da informação do bitstream passa por um decodificador de entropia, que fornece coeficientes quantizados de transformada. Esta codificação de entropia pode ser dos tipos CAVLC (Context-adaptive Variable Length Codging) ou CABAC (Context-adaptive Binary Arithmetic Coding). O CABAC apresenta maior taxa de compressão, mas por outro lado tem custo computacional bem mais elevado. Informações mais detalhadas sobre a codificação de entropia são dadas nas seções 3.3 e 3.4.

2.3.6 Mudança de escala e transformada inversas

A decodificação de entropia recupera coeficientes quantizados no domínio frequência, que passam então por uma etapa de mudança de escala. A transformada inversa é então feita sobre os coeficientes, recuperando os resíduos do bloco atual.

A transformada usada é baseada na transformada DCT (Direct Cosine Transform), que foi alterada para usar apenas somas e deslocamentos de bit, o que reduz significativamente sua complexidade computacional. Para os coeficientes DC dos blocos de luminância de macroblocos codificados com predição intra-quadro de tamanho 16x16 pixels e dos blocos de crominância, antes da DCT inversa é aplicada uma transformada Hadamard inversa, visto que tais coeficientes passaram por uma etapa de transformada Hadamard no codificador, a fim de reduzir ainda mais a correlação entre estes coeficientes.

2.3.7 Filtro de deblocagem

Antes de ser exibida e armazenada como referência para predição inter-quadro, a imagem passa por um filtro de deblocagem, que ameniza o efeito de distorção de blocos gerado pelas quantização das transformadas. Diferentemente de padrões de codificação mais antigos, o H.264 deixa o filtro no laço de decodificação, ou seja, a imagem é filtrada antes de ser armazenada como referência para predição inter-quadros. Este fato melhora bastante a qualidade final da imagem e evita que artefatos gerados na imagem dificultem a busca do codificador por um bloco semelhante nas referências.

2.3.8 Perfis e níveis

O padrão H.264 define perfis e níveis para limitar as funcionalidades usadas na codificação de um determinado vídeo, bem como os requisitos de desempenho e memória para a decodificação do mesmo.

Os perfis do H.264 previstos para o SBTVD são:

- Baseline – perfil mais simplificado, onde não é permitido o uso de bi-predição, CABAC e entrelaçamento. Usado na transmissão de vídeo para dispositivos móveis. Conta com mecanismos de redundância e resiliência a erros;

- Main – perfil intermediário, onde é permitido o uso de bi-predição, CABAC e entrelaçamento;

- High – perfil mais completo, onde além das funcionalidades previstas para o perfil Main, há a possibilidade de usar transformadas de tamanho 8x8 e matrizes de quantização customizadas.

Os níveis definem o desempenho mínimo que o decodificador de vídeo precisa ter para suportar aquela sequência de vídeo. A escolha do nível determina, entre outros:

- resolução máxima da imagem;
- taxa de exibição máxima;
- valores máximos para vetores de movimento;
- tamanho do buffer de imagens decodificadas para formar as listas de referência para predição inter-quadros.

3 MÓDULO PARSER NO DECODIFICADOR H.264

Num terminal de acesso de TV Digital, o circuito de sintonia e demodulação recebe os dados da emissora através da antena. Depois, um demultiplexador separa as informações para os respectivos decodificadores de vídeo, áudio e dados.

No projeto do terminal de acesso que a Rede H.264 desenvolve, o fluxo de dados de vídeo é recebido pelo decodificador de vídeo através de uma interface DVB-SPI. A frequência de operação da interface é de 27MHz, recebendo 1 byte a cada 8 ciclos.

A função do parser no decodificador é identificar os elementos recebidos no fluxo de bits de entrada, repassando-os ao módulo responsável por sua interpretação. Os elementos sintáticos são representados no fluxo de entrada codificado sob diferentes formas, com descritores de acordo com o tipo de parâmetro que se está recebendo. Na seção 7.2 das normas da ITU-T [4] são apresentados os descritores dos tipos de dados a serem lidos, que podem ser dos seguintes tipos:

- ae(v) – elemento codificado com entropia CABAC;
- ce(v) – elemento codificado com entropia CAVLC;
- me(v) – elemento Exp-Golomb mapeado;
- se(v) – elemento Exp-Golomb com sinal;
- te(v) – elemento Exp-Golomb truncado;
- u(n) – elemento com n bits;
- ue(v) – elemento Exp-Golomb sem sinal.

Na lista anterior, *v* representa um elemento com número variável de bits, e *n* um elemento com número fixo de bits.

3.1 Unidades de NAL

No padrão de codificação H.264, os dados são encapsulados em unidades de NAL (Network Abstraction Layer). Na seção 7.3 das normas da ITU-T [4] podemos encontrar a sintaxe dos pacotes NAL. Existe um delimitador que identifica as fronteiras entre unidades. Tal delimitador é caracterizado por uma sequência de três bytes, de valor *0x000001*. O padrão de codificação prevê mecanismos para impedir que esta sequência apareça nos dados transmitidos pela unidade, o que acarretaria na falsa identificação do seu fim.

Após o identificador, cada unidade de NAL tem um cabeçalho com tamanho de um byte, indicando o tipo da unidade. O tipo indica que estrutura está contida na carga útil da unidade (*payload*). As subseções seguintes apresentam os tipos de unidades usadas neste trabalho.

3.1.1 SPS

A fim de identificar corretamente os dados de imagem, o decodificador recebe conjuntos de parâmetros que são essenciais à correta execução dos algoritmos de decodificação. Esses parâmetros incluem desde informações básicas como as dimensões da imagem até operações complexas de reordenamento de listas de referência para predição inter-quadros.

Os pacotes SPS (Sequence Parameter Set) são identificados pelo tipo de NAL 7 e contêm informações sobre a sequência de vídeo que se está decodificando. Por conter propriedades da imagem como, por exemplo, a altura e a largura, são necessários para a decodificação das imagens. Alguns dos parâmetros mais relevantes contidos no SPS são:

- Perfil e nível do vídeo codificado – as normas do SBTVD [5] determinam compatibilidade com os perfis Baseline, Main e High, que limitam os mecanismos de codificação presentes no vídeo. O nível máximo permitido é 4.0, que determina requisitos de memória para listas de referência, limita a área de busca dos vetores de movimento, a resolução, a taxa de exibição, etc;

- Altura e largura da imagem;

- Número máximo de imagens de referência nas listas para predição inter-quadro;

- Uso de entrelaçamento e entrelaçamento adaptativo por macrobloco (MBAFF).

A parte final da carga útil do pacote SPS destina-se à transmissão opcional dos parâmetros VUI (Video Usability Information), que define informações para a exibição do vídeo após a decodificação, como razão de aspecto e informações de temporização.

3.1.2 PPS

Os pacotes PPS (Picture Parameter Set) são identificados pelo tipo de NAL 8 e contêm informações de imagem para um conjunto de imagens decodificadas. Assim como os pacotes SPS, sua existência é requisito para a decodificação do vídeo. Seus parâmetros mais relevantes são:

- Modo de codificação de entropia – CAVLC ou CABAC;
- Parâmetro de quantização dos coeficientes de transformada para geração de resíduos;
- Número de imagens ativas nas listas 0 e 1 para predição inter-quadro.

3.1.3 Slices

Cada imagem gerada pelo decodificador pode ser dividida em *slices*, de acordo com a codificação que foi feita. Os *slices* são um conjunto de macroblocos – blocos de 16x16 pixels – que formam uma região da imagem decodificada, podendo constituir toda a imagem (ou campo para vídeo entrelaçado). A técnica de divisão da imagem em slices é utilizada principalmente para conter possíveis erros na transmissão do sinal a apenas uma parte da imagem final vista pelo espectador. Nas transmissões de alta definição das emissoras brasileiras, pode-se constatar que geralmente cada imagem é formada por seis slices.

Cada slice é identificado pelos tipo de NAL 1 ou 5 e contém as informações necessárias à decodificação da imagem contida naquele pacote. O slice começa com um cabeçalho, denominado *slice header*. Alguns de seus parâmetros mais importantes são:

- Posição do primeiro macrobloco do slice;
- Tipo de slice – determina os tipos de predição permitidas no slice;
- Uso de entrelaçamento e identificação de campo superior ou inferior;
- Operações de reordenamento das listas de imagens de referência;

- Variação do parâmetro de quantização no slice;
- Operações de controle sobre o filtro de deblocagem.

Após o cabeçalho, são interpretados os dados de imagem, transmitidos para cada macrobloco. Tais dados incluem:

- Indicação do número de macroblocos do tipo Skip – para estes macroblocos não são transmitidos modos de predição, vetores de movimentos e resíduos.
- Indicação do modo quadro ou campo para o caso de MBAFF;
- Tipo de macrobloco – indica uso de predição intra-quadro ou inter-quadro. No caso de predição inter-quadro, indica também o particionamento de macrobloco e as listas de referência utilizadas;
- Tipo de sub-macrobloco, quando o macrobloco for particionado em quatro partições de 8x8 pixels;
- Modo de predição intra-quadro, se o tipo de macrobloco requer;
- Imagem utilizada das listas de referência 0 e 1, se o tipo de macrobloco requer;
- Vetores de movimento para cada partição de macrobloco ou partição de sub-macrobloco, conforme indicação do tipo de macrobloco e de sub-macrobloco;
- Variável *coded_block_pattern* – indica a presença de resíduos para cada partição de macrobloco (tamanho 8x8 pixels);
- Variação do parâmetro de quantização no macrobloco;
- Coeficientes quantizados para geração de resíduos, codificados conforme tipo de codificação de entropia identificado no PPS.

3.2 Codificação Exp-Golomb

A codificação Exp-Golomb utiliza códigos binários de comprimento variável para representar os elementos codificados. Códigos mais curtos são atribuídos aos valores menores. Os elementos marcados pelos descritores me(v), se(v), te(v) e ue(v) utilizam esta codificação.

O processo de decodificação de um elemento Exp-Golomb segue o algoritmo descrito pelo pseudocódigo abaixo. Iniciando da posição atual do bitstream, conta-se quantos bits '0' estão presentes antes do primeiro '1', que é descartado. O mesmo número de bits obtido anteriormente é lido após o primeiro '1'.

```
leadingZeroBits = -1;
for(b=0; !b; leadingZeroBits++)
   b = read_bits(1);
```

A variável *codeNum* é determinada pela equação 1:

$$codeNum = 2^{leadingZeroBits} - 1 + read\_bits(leadingZeroBits) \tag{1}$$

As faixas de valores de *codeNum* para cada número de bits de representação são mostradas na tabela 1.

Tabela 1: Faixas de *codeNum*

Formato do código	Faixa de *codeNum*
1	0
0 1 x_0	1-2
0 0 1 x_1 x_0	3-6
0 0 0 1 x_2 x_1 x_0	7-14
0 0 0 0 1 x_3 x_2 x_1 x_0	15-30
...	...

Se o elemento sintático foi codificado como ue(v), seu valor é igual ao de $codeNum$.

Para os elementos te(v) deve-se conhecer o tamanho máximo previsto para aquele elemento no momento de sua decodificação; se tal valor for maior que 1, o elemento é lido como se fosse do tipo ue(v); caso contrário, a inversão do valor lido em um único bit determina seu valor.

Caso seja do tipo me(v), seu valor será determinado pelas tabelas da seção 9.1.2 das normas da ITU-T [4], indexadas pelo valor de $codeNum$. Este tipo de codificação é utilizado pelo elemento sintático *coded_block_pattern*.

Se a codificação for do tipo se(v), seu valor é associado ao resultado da equação 2:

$$se = (-1)^{codeNum+1} Ceil(codeNum/2) \tag{2}$$

3.3 Codificação de entropia CAVLC

A codificação de entropia CAVLC é utilizada para compactar os coeficientes quantizados usados para geração dos resíduos somados ao resultado da predição para recuperar a imagem decodificada. Este modo de codificação de entropia está presente em todos os perfis do H.264.

Seu decodificador, o CAVLD, tem tabelas indexadas pelos dados de entrada contendo os elementos sintáticos intermediários para a decodificação de um bloco 4x4 de coeficientes quantizados. Entretanto, o decodificador escolhe qual tabela utilizar de maneira adaptativa, baseado no número de coeficientes não nulos dos blocos vizinhos ao bloco atual.

A decodificação de um bloco envolve contextos que determinam:

- o número de coeficientes não nulos dos bloco;
- o número de coeficientes de valor absoluto unitário no fim de uma varredura zig-zag do bloco;
- o sinal dos coeficientes unitários;
- o número de coeficientes nulos antes do último coeficiente não nulo.

Após a identificação destes contextos nas tabelas, laços de controle são capazes de inserir os coeficientes nulos na posição correta da varredura zig-zag do bloco.

3.4 Codificação de entropia CABAC

Assim como o CAVLC, o CABAC também é empregado para compactação de coeficientes quantizados. Além disso, quando usado na codificação do vídeo, o CABAC é usado para codificar todos os elementos sintáticos do slice data. Este método de codificação de entropia é utilizado apenas no perfis Main e High do H.264, não podendo ser utilizado nas transmissões *one-seg*.

O CABAD, decodificador de elementos codificados em CABAC, deve atualizar seus contextos internos a cada *bin* decodificado. Um *bin* é uma decisão binária feita durante a decodificação de um elemento. A adaptação dos contextos segue um modelo de probabilidade de ocorrência de códigos. Após a decodificação de uma sequência de bins, é feito um processo de anti-binarização para obter o valor do elemento sintático.

O CABAD não é utilizado neste trabalho, visto que atualmente está em processo de desenvolvimento no projeto do decodificador H.264.

4 METODOLOGIA E FERRAMENTAS UTILIZADAS

Durante o desenvolvimento do parser, seus sub-módulos são projetados e implementados um a um e então integrados ao restante do parser.

Para validação, é utilizado um módulo VHDL que engloba uma instância do módulo que se deseja testar. Este módulo, chamado *testbench*, fornece estímulos ao módulo sob teste e registra as saídas geradas. As saídas são então comparadas aos resultados esperados para verificar a correta interpretação dos estímulos.

Para a validação do parser, havia a possibilidade de se criar um testbench específico para cada módulo. Esta alternativa seria pouco adequada no contexto de projeto atual, visto que sem a utilização de módulos auxiliares que gerenciam o consumo de dados da entrada do parser, os sinais de entrada para cada módulo seriam manualmente criados pelo projetista e serviriam para um único bitstream.

Portanto, decidiu-se utilizar um testbench global ao parser integrado, herdado do projeto anterior de um parser mínimo. Para a integração, basta alterar a máquina de estados do controle global do parser para usar o novo módulo integrado, bem como do módulo que identifica unidades de NAL.

Depois de validado, o projeto pode então ser integrado ao restante do decodificador de vídeo, onde então passa mais uma vez por um processo de validação. Após validado, o decodificador pode ser sintetizado para o FPGA da placa de desenvolvimento utilizada.

Cabe notar que há a possibilidade de mudar a plataforma de FPGA durante diferentes etapas do projeto, visto que o dispositivo usado atualmente não tem elementos lógicos suficientes para implementar o decodificador de vídeo completo. Além disso, na conclusão da implementação do decodificador, o grupo de pesquisa do LaPSI precisa especificar um ASIC para posterior projeto e fabricação. Portanto, os arquivos fonte VHDL devem ser escritos utilizando apenas as bibliotecas padrão do VHDL, para que o projeto não seja

dependente da plataforma de desenvolvimento, acrescentando um diferencial de portabilidade ao resultado final da equipe.

4.1 Normas

4.1.1 ITU-T

Publicado pela ITU-T, o conjunto de recomendações é a documentação oficial sobre a codificação H.264. No projeto, usamos a edição de março de 2005, conforme recomendado pelas normas do SBTVD. Nesse documento, estão especificadas as operações do decodificador de vídeo, descrevendo os algoritmos necessários para a identificação dos elementos sintáticos, sua interpretação, faixa permitida de valores, geração de variáveis derivadas e outros.

4.1.2 ABNT

As normas específicas para o SBTVD foram publicadas pela ABNT em uma série de documentos que regulamentam todo o sistema. Para a codificação e decodificação de vídeo, a norma usada é a NBR 15602-1 [5].

4.2 Simulação VHDL

Para simular os testbenches foi utilizado o GHDL, simulador VHDL de código aberto distribuído sob a licença GNU GPL. O simulador utiliza a tecnologia do compilador GCC, compilando os arquivos VHDL do projeto que se deseja simular, avaliando as dependências entre eles e gerando um arquivo executável, que executa o testbench do módulo sob teste. Após a geração do ambiente de trabalho, que inclui todos os arquivos fonte utilizados, pode-se usar o GHDL para gerar um arquivo *Makefile* que pode ser usado para facilmente atualizar o executável após a alteração de um ou mais arquivos fonte.

Para visualização das formas de onda resultantes, o GHDL pode exportar os valores dos sinais simulados em um arquivo do formato GHW (GHDL Waveform), formato binário definido pelo desenvolvedor do simulador GHDL. Para visualizar este arquivo, pode-se usar o software *GTKWave*, também de código aberto. Alternativamente, pode-se exportar os dados no formato VCD (Value Change Dump), formato aberto definido pela linguagem Verilog suportado pela maioria dos visualizadores de forma de onda.

4.3 Geração de estímulos

Para alimentar a simulação do módulo parser, precisamos obter estímulos a serem usados. Como este módulo é responsável por interpretar o bitstream de entrada, conforme explicado no capítulo 3, o estímulo necessário é um bitstream válido de um vídeo codificado em H.264. Para obter este arquivo, duas ferramentas foram utilizadas.

4.3.1 Codificador JM

O software de referência para o padrão H.264/AVC é o JM [6], fornecido pela ITU-T. Este software é escrito em linguagem C e é composto por um codificador e um decodificador de vídeo. Para a geração de bitstreams de teste foram utilizadas sequências de vídeo no formato YUV, sem compactação. Estas sequências estão disponíveis na internet e são bem estabelecidas na literatura como teste padrão para o comportamento de codificadores e decodificadores de vídeo. O codificador é então configurado para utilizar as funcionalidades requeridas na verificação, de acordo com as normas brasileiras da ABNT. As sequências são então codificadas e o arquivo *.264* resultante serve como entrada para o módulo parser nas simulações.

4.3.2 Captura de bitstream das emissoras

A fim de testar o decodificador em uma situação real de decodificação, foi desenvolvido um sistema de captura de bitstream das emissoras locais de TV. Para tanto, foi utilizado um receptor USB de TV digital da PixelView. Este receptor é ligado a um computador com sistema operacional Linux, de onde aplicativos dos pacotes *video4linux* [7] e *dvb-apps* [8] são capazes de encontrar os canais disponíveis, fazer a sintonia e capturar o bitstream no nível de transporte, ou seja, com todas as informações de vídeo, áudio e dados. Um demultiplexador do pacote *dvb-apps* extrai as informações de vídeo e salva em um arquivo *.264*.

4.3.3 Geração de resultados esperados

Para a geração dos resultados esperados pelo módulo parser, foi utilizado o software PRH.264 [9, 10]. Este software foi desenvolvido no LaPSI e se trata de um decodificador H.264 implementado em C, com a preocupação de manter a estrutura do software modularizada de modo a imitar uma arquitetura de hardware. Com o desenvolvimento

deste software, criou-se na equipe um conhecimento mais profundo de todas as etapas de decodificação.

Para gerenciar sua interface gráfica, o PRH.264 utiliza as bibliotecas WxWidgets [11] e SDL [12], ambas de código aberto e multi-plataforma. A interface do PRH.264 é mostrada na figura 7.

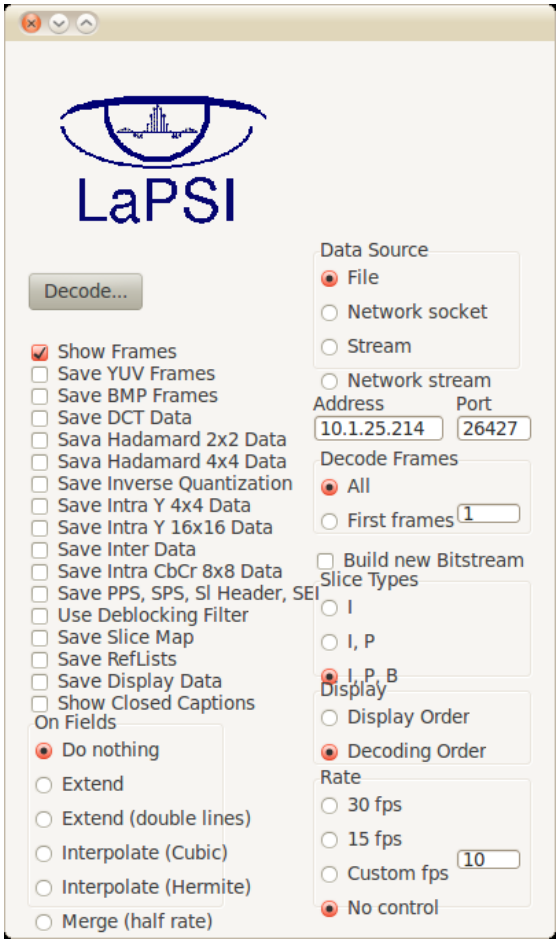

Figura 7: Interface gráfica do PRH.264

O PRH.264 é capaz de acessar as entradas e saídas dos vários módulos intermediários da decodificação, de modo que pode ser então utilizado para extrair os resultados esperados do módulo parser para este trabalho. Dado um vídeo codificado, o PRH.264 salva os parâmetros interpretados em um arquivo texto, que pode facilmente ser utilizado para comparar com resultados de simulação do módulo de hardware que se está projetando. Tal comparação pode ser feita manualmente ou de modo automático, inserindo no test-bench estruturas que convertem o arquivo texto gerado pelo PRH.264 para um formato utilizável para comparação na simulação.

Além do PRH.264, o arquivo *trace* gerado pelo decodificador JM [6] foi utilizado em

algumas situações específicas. Este arquivo registra todos os elementos identificados pelo parser deste software, mostrando a sequência de bits usada para a decodificação de cada elemento, bem como o valor interpretado após sua decodificação.

4.4 Síntese VHDL

Para síntese dos arquivos VHDL para o FPGA da placa de desenvolvimento, foi utilizado o Xilinx ISE, na versão 10.1. Entre outras alternativas, a ferramenta foi escolhida pois o LaPSI possui licensa para utilização da mesma. Apesar de a versão 10.1 não ser a última versão deste software, que atualmente está na versão 11.5, ela foi escolhida por ser a mais recente a oferecer suporte à placa de desenvolvimento usada no projeto.

4.5 Placa de prototipação

A placa de prototipação utilizada neste trabalho é a Digilent XUPV2P, mostrada na figura 8. A placa foi escolhida tanto por sua disponibilidade para uso no LaPSI quanto pela experiência de uso dos pesquisadores com esta placa. A placa contém um FPGA Xilinx Virtex II-Pro XC2VP30 e diversos periféricos importantes para o desenvolvimento do decodificador, como saída VGA de vídeo, interface de comunicação serial e conexão Ethernet.

Figura 8: Placa de prototipação Digilent XUPV2P

Entretanto, já se sabe que com a integração dos diversos módulos do decodificador de vídeo à implementação atual, este FPGA não comportará o decodificador inteiro. Para prototipar o decodificador após a integração do módulo de predição inter-quadros, uma nova placa de prototipação será utilizada. A placa escolhida é a Digilent XUPV5, que

contém um FPGA Xilinx Virtex-5 XC5VLX110T. Além dos periféricos citados para a Digilent XUPV2P, esta placa inclui uma saída de vídeo DVI, que suporta as resoluções necessárias para exibição de vídeo em alta definição.

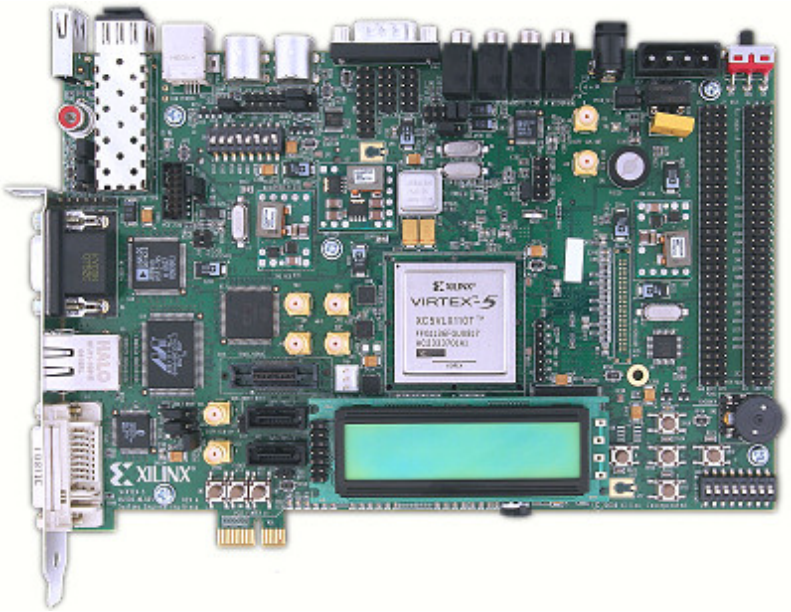

Figura 9: Placa de prototipação Digilent XUPV5

5 ARQUITETURA PROPOSTA

A arquitetura desenvolvida neste trabalho para o projeto do módulo parser é baseada na arquitetura herdada de um projeto anterior de um parser mínimo [13, 14], que foi utilizado para compor um protótipo inicial do decodificador de vídeo. Alguns dos módulos herdados foram reaproveitados em sua totalidade, enquanto outros foram completamente reescritos ou sofreram pequenas alterações para serem compatíveis com os novos módulos. A figura 10 apresenta a estrutura geral da arquitetura proposta para o módulo parser.

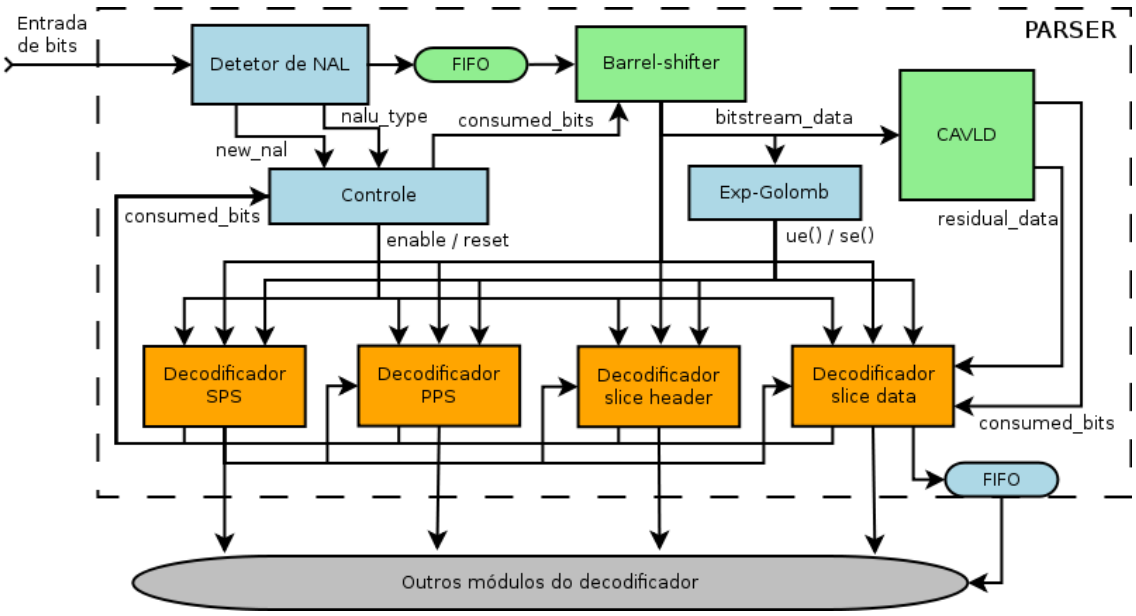

Figura 10: Estrutura geral da implementação do módulo parser

Os módulos representados em laranja na figura 10 foram completamente desenvolvidos neste trabalho e serão detalhados nas seções 5.2 a 5.5.

Os módulos representados em verde na figura 10 foram reaproveitados neste trabalho. Tais módulos implementam a FIFO de entrada e sua interface de controle, o *barrel-shifter* utilizado para atualizar a entrada dos módulos de decodificação de acordo com o consumo

dos bits da entrada, bem como o módulo de decodificação de entropia CAVLC.

Os módulos de controle, de detecção de nova unidade de NAL e decodificador de elementos Exp-Golomb – representado em azul na figura 10 – tiveram pequenas alterações em suas descrições, a fim de controlar e fornecer dados de entrada necessários para o funcionamento dos blocos desenvolvidos neste trabalho.

5.1 Estrutura geral das máquinas de estado

A implementação dos módulos decodificadores de SPS, PPS, slice header e slice data é feita com máquinas de estados cuja forma geral de descrição é bastante semelhante.

Em todas estas máquinas, cada ciclo de máquina leva quatro ciclos de relógio. Em cada estado é feita a leitura de um parâmetro do bitstream ou são alterados registradores responsáveis pelo armazenamento de sinais que determinam tomadas de decisões de controle de fluxo. Um sinal controla o consumo de dados de cada estado, selecionando se o consumo vem do CAVLD, do decodificador de Exp-Golomb ou é representado por um número fixo de bits. Este sinal é registrado e enviado ao módulo de controle do parser.

A figura 11 mostra o temporizador de estado, o sinal de consumo de bits e algumas transições de estado.

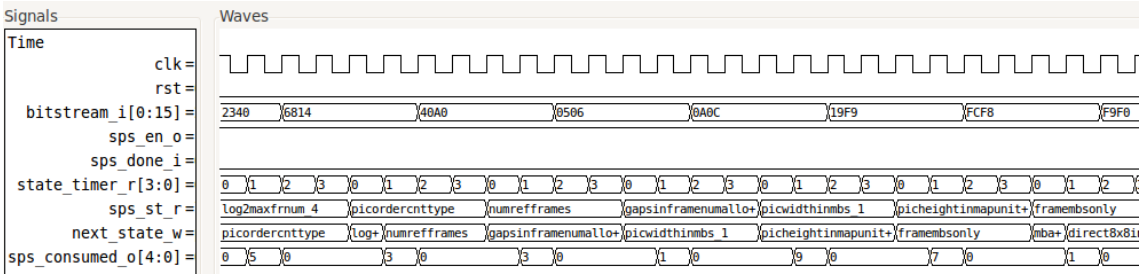

Figura 11: Sinais de temporização, de consumo de bits da entrada, de estado atual e de próximo estado

5.2 Decodificador de SPS

Para a decodificação dos pacotes de SPS, foi desenvolvido um módulo VHDL cuja interface é mostrada na figura 12. Este módulo tem acesso direto ao bitstream de entrada e também às saídas do módulo de decodificação de elementos codificados no formato Exp-Golomb.

Os elementos decodificados que são necessários diretamente em outros módulos do

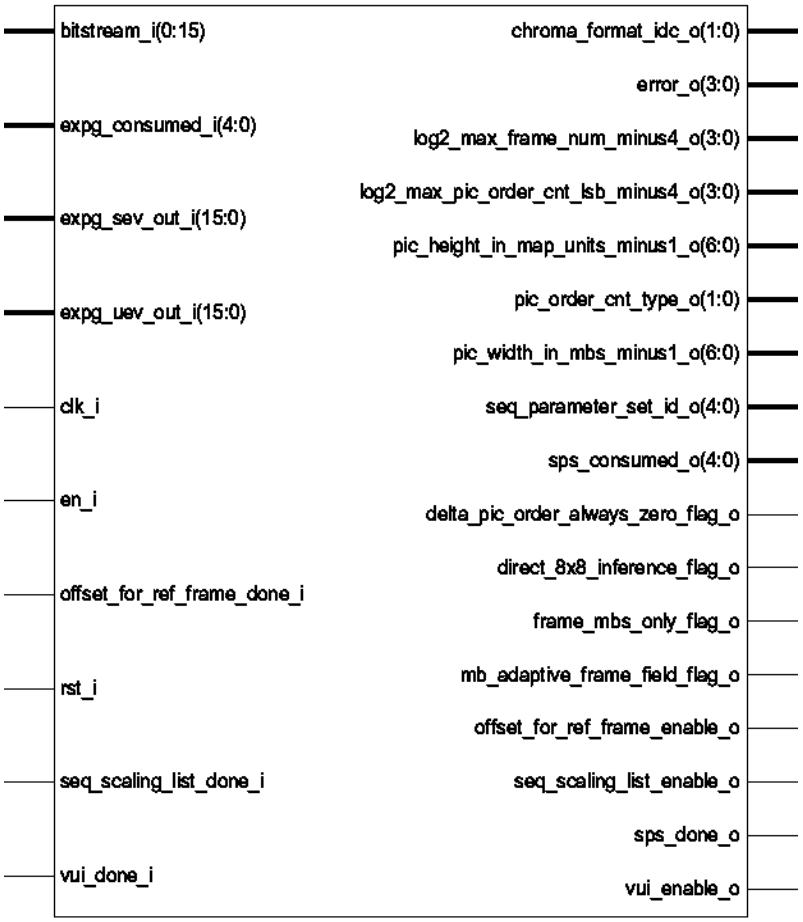

Figura 12: Interfaces do módulo desenvolvido para o decodificador de SPS

parser ou do restante do decodificador nesta etapa de projeto têm saída declarada na entidade da arquitetura. Os outros elementos são registrados internamente na descrição, e por não serem utilizados posteriormente são eliminados pela ferramenta de síntese.

A sintaxe completa dos dados de um pacote SPS podem ser encontradas na seção 7.3.2.1 das normas da ITU-T [4]. A máquina de estados implementada é mostrada na figura 13. Vale notar que esta máquina de estados comanda outras máquinas menores, voltadas à decodificação de sequências específicas de elementos, como os pacotes VUI ou as listas de *seq_scaling_list* ou *offset_for_ref_frame*. Na figura 13, as interações com tais máquinas são identificadas como transições tracejadas de estado.

As limitações da implementação deste módulo residem no fato que essas "máquinas-filhas" não foram implementadas no desenvolvimento deste trabalho. O módulo tranca em um estado, indicando um erro pela presença dos elementos decodificados pelas referidas máquinas.

5.3 Decodificador de PPS

A interface da arquitetura desenvolvida para a decodificação dos pacotes de PPS é mostrada na figura 14. Assim como no módulo para os pacotes SPS, este módulo tem acesso direto ao bitstream de entrada e também às saídas do módulo de decodificação de elementos codificados no formato Exp-Golomb. A seleção dos sinais que têm saída dedicada na arquitetura segue os mesmos critérios que para o módulo decodificador de SPS.

A sintaxe completa dos dados de um pacote PPS podem ser encontradas na seção 7.3.2.2 das normas da ITU-T [4]. A máquina de estados implementada é mostrada na figura 15. Esta máquina de estados comanda outra máquina menor, representada pelo estado tracejado na figura, voltada à decodificação de *pic_scaling_list*. Assim como no decodificador de SPS, a "máquina-filha" não foi implementada.

5.4 Decodificador de slice header

Para a identificação dos parâmetros recebidos no slice header, foi desenvolvido o módulo cuja interface é mostrada na figura 16. Novamente, a escolha dos sinais que têm saída na arquitetura segue os mesmos critérios de uso explicados anteriormente.

A sintaxe completa dos dados contidos no slice header podem ser encontradas na seção 7.3.3 das normas da ITU-T [4]. A máquina de estados implementada é mostrada na figura 17. Note a existência de vários estados de passagem, denominados *chk#*, que servem apenas para testar as variáveis que decidem se a máquina deve ou não entrar no próximo estado.

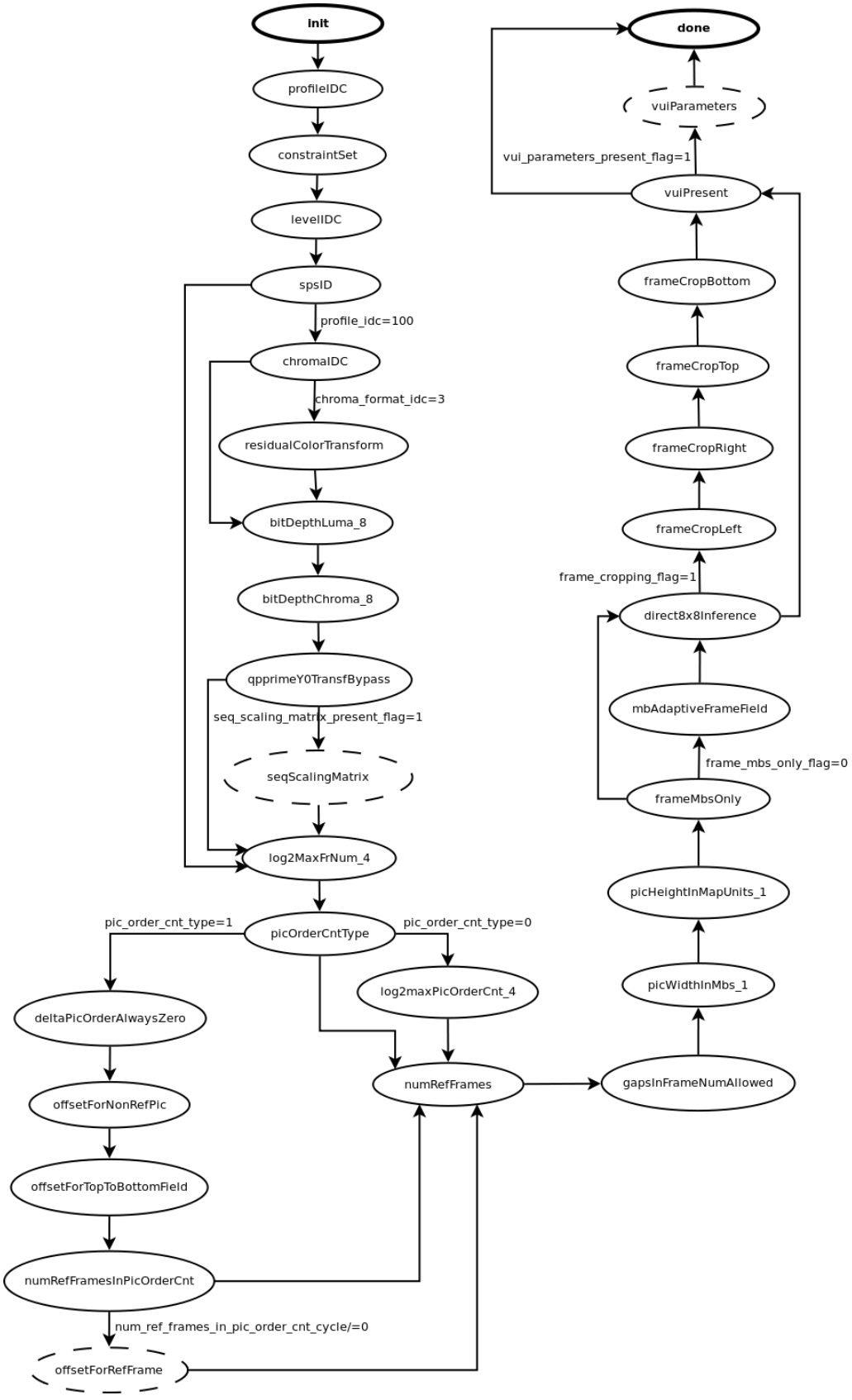

Figura 13: Diagrama de estados do decodificador de SPS

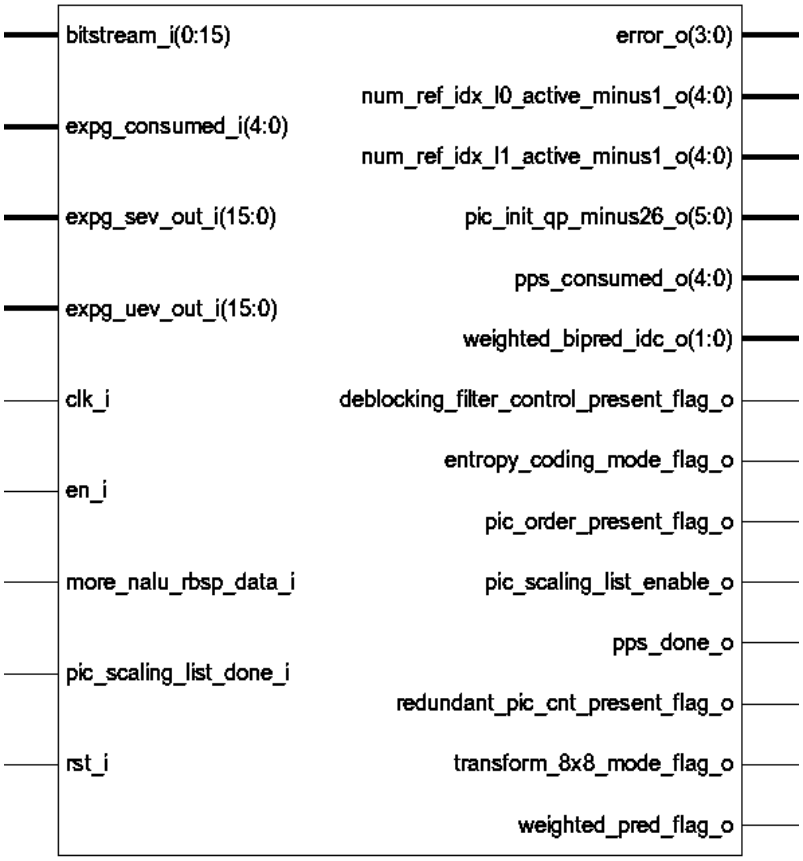

Figura 14: Interfaces do módulo desenvolvido para o decodificador de PPS

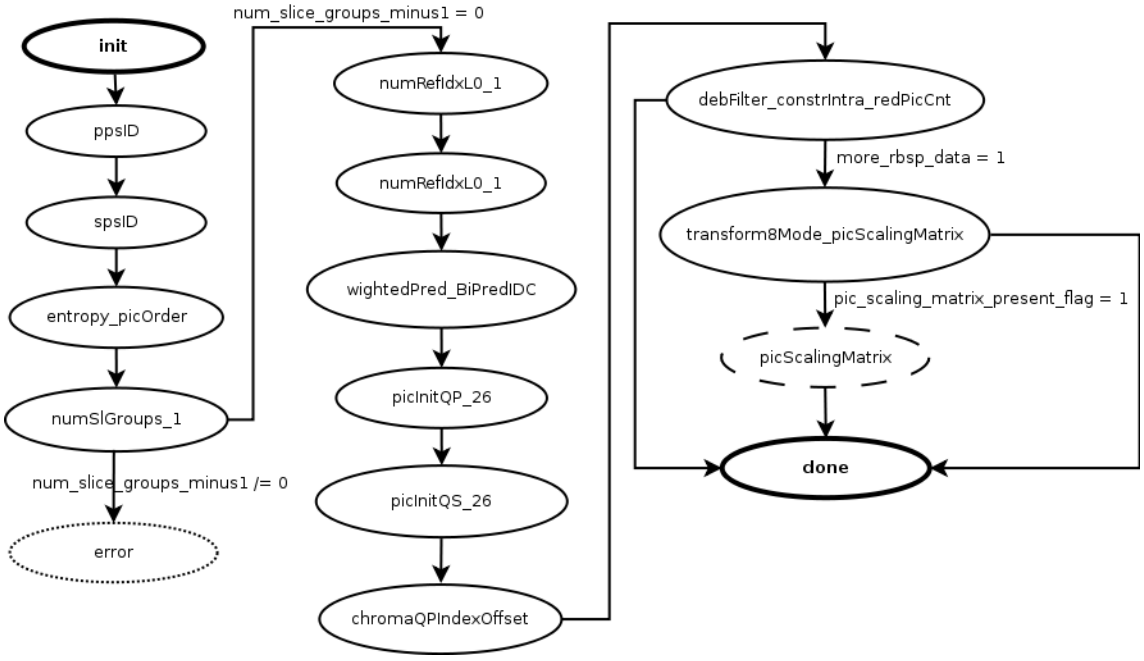

Figura 15: Diagrama de estados do decodificador de PPS

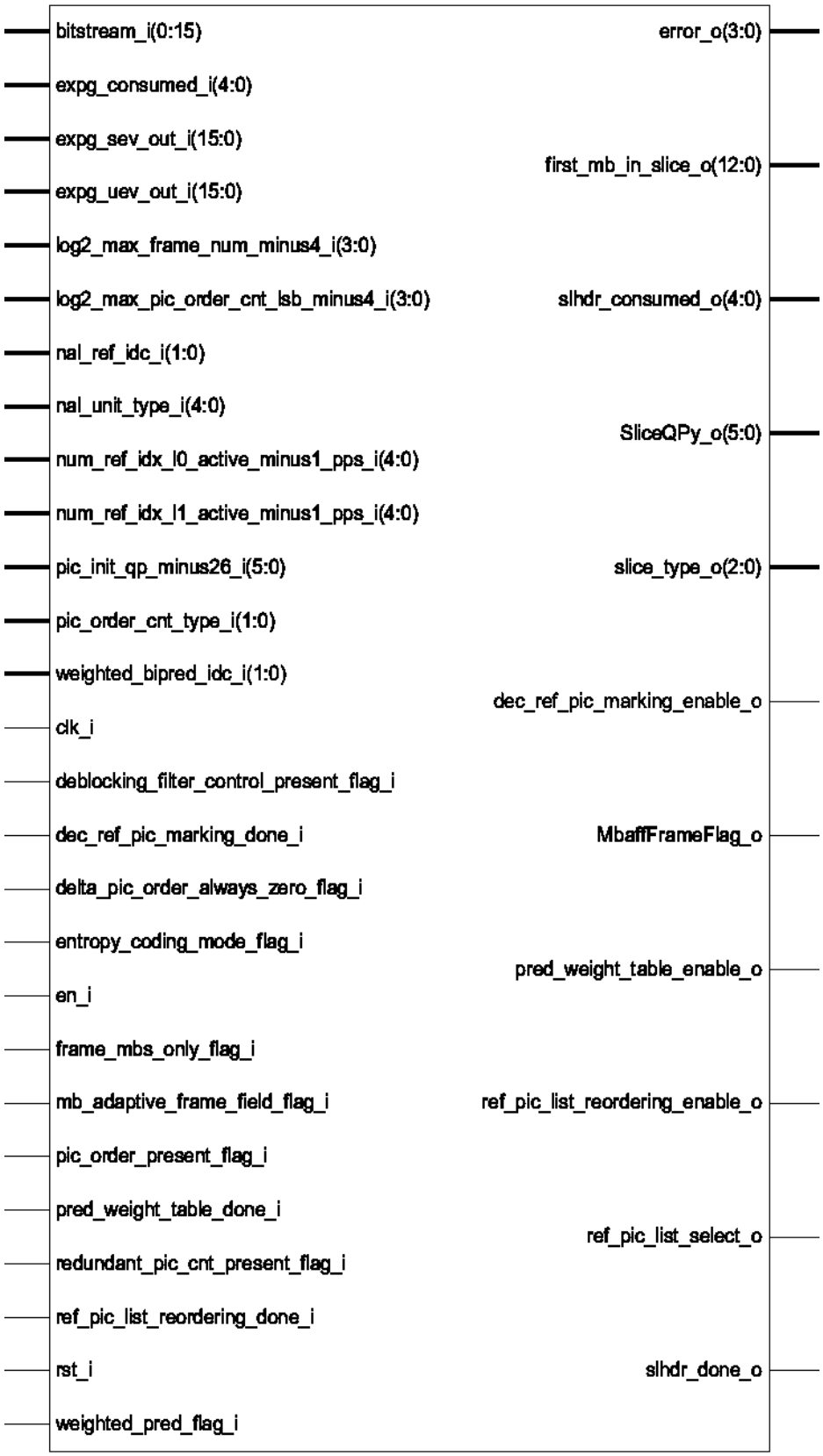

Figura 16: Interfaces do módulo desenvolvido para o decodificador de slice header

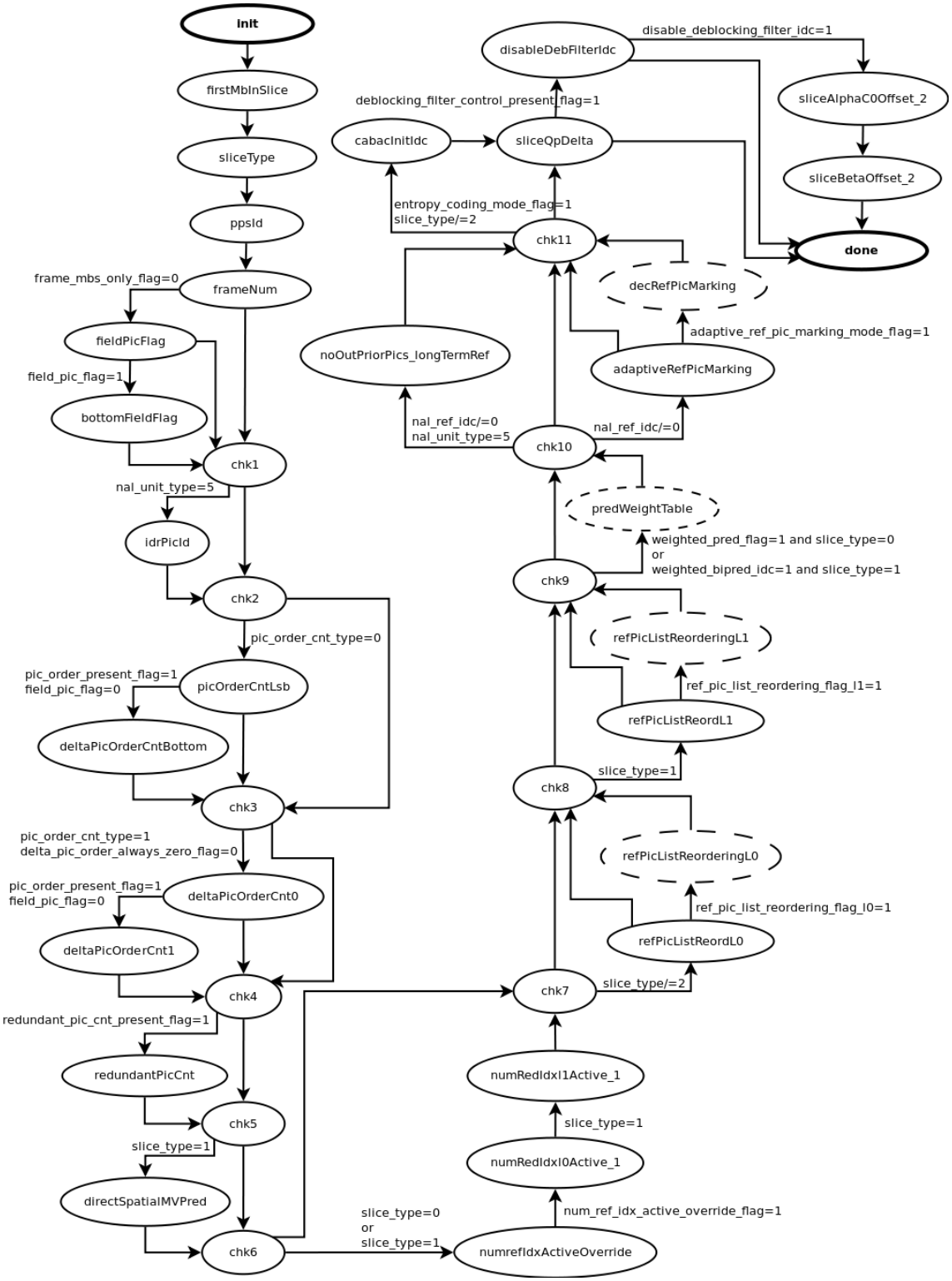

Figura 17: Diagrama de estados do decodificador de slice header

5.5 Decodificador de slice data

A interface do módulo desenvolvido para a identificação dos elementos de slice data é mostrada na figura 18.

A sintaxe completa dos dados de slice data podem ser encontradas na seção 7.3.4 das normas da ITU-T [4]. A máquina de estados implementada é mostrada na figura 19. Assim como na máquina de estados destinada ao decodificador de slice header, existem vários estados de passagem.

Em contraste com as outras máquinas de estado implementadas, esta máquina tem vários laços de execução. Estes laços executam varreduras para obtenção dos modos de predição intra, dos vetores de movimento e das imagens de referência, de acordo com o tipo de macrobloco, de sub-macrobloco e outros parâmetros de configuração.

Neste módulo são utilizadas várias tabelas, usadas para determinar o tipo de macrobloco em função do tipo de slice, além de características do macrobloco como número de partições e tipo de predição em função de seu tipo. Tabelas são usadas também para derivar o parâmetro *coded_block_pattern*, que identifica as partições do macrobloco para as quais existe resíduo codificado no bitstream.

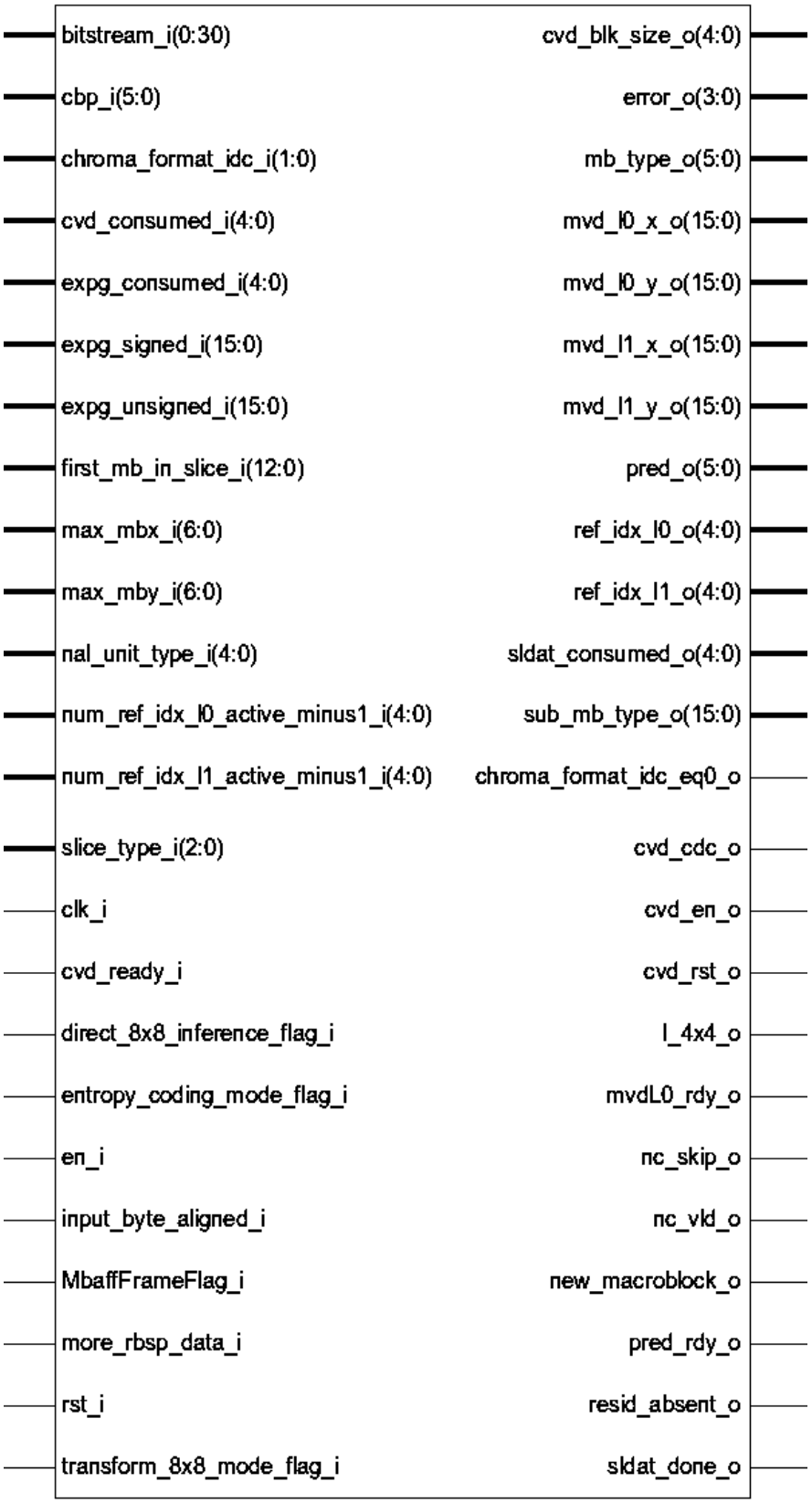

Figura 18: Interfaces do módulo desenvolvido para o decodificador de slice data

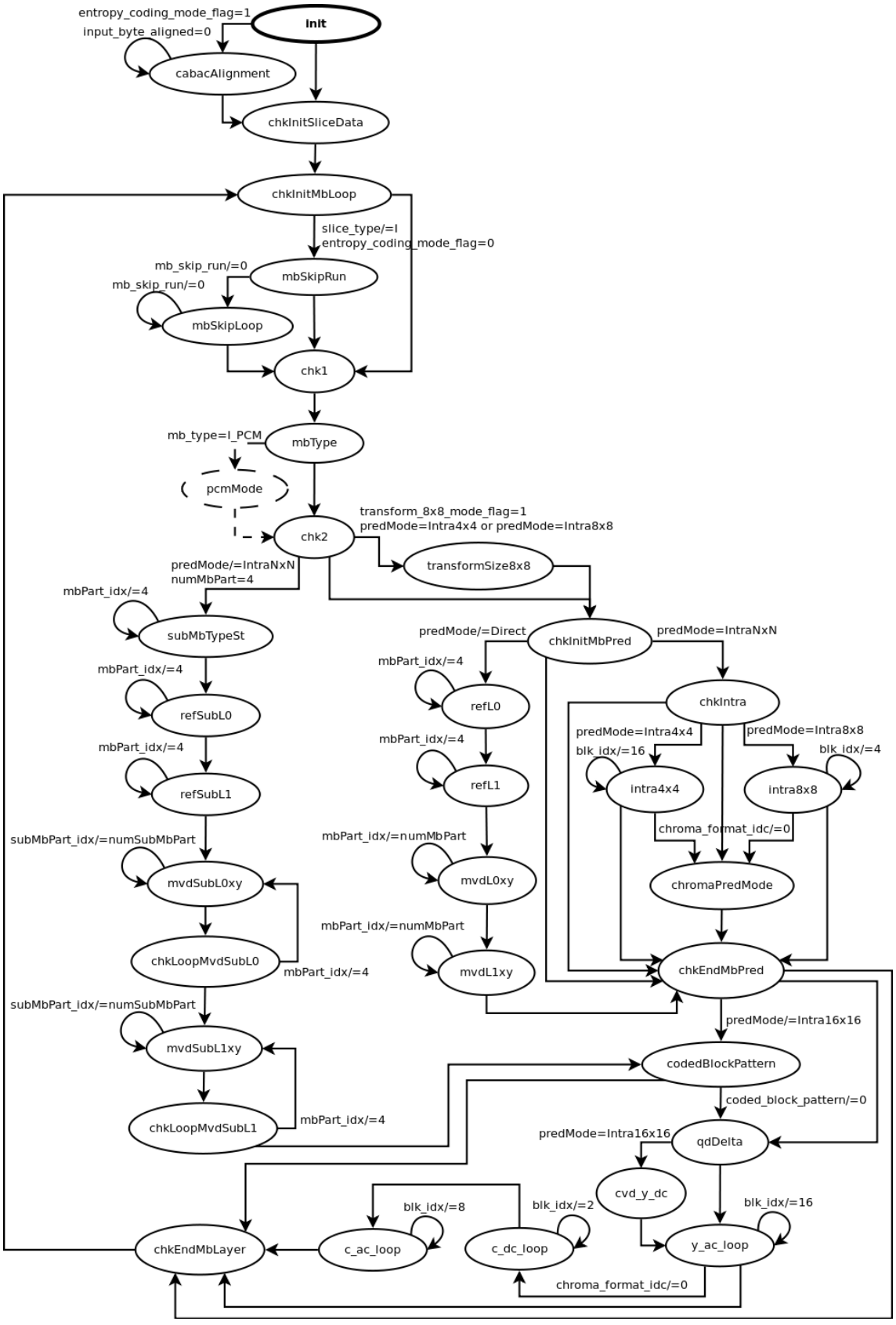

Figura 19: Diagrama de estados do decodificador de slice data

5.6 Módulos herdados

5.6.1 Controle do parser

A função do módulo de controle no parser é habilitar seus módulos de decodificação – de tipo de NAL, SPS, PPS, slice header ou slice data – de acordo com o fluxo previsto pela codificação H.264. Está sob sua responsabilidade coletar o consumo de cada módulo de decodificação, comandar o deslocamento dos dados de entrada através do barrel-shifter e solicitar mais dados ao controlador da FIFO de entrada do decodificador de vídeo. Também recebe informações dos controladores das FIFOs de saída do parser, interrompendo os módulos de decodificação se alguma das FIFOs está cheia.

5.6.2 Barrel-shifter

O barrel-shifter implementado no parser tem por objetivo disponibilizar os bits corretos do bitstream aos módulos que vão consumi-lo. Para tanto, deve deslocar os dados registrados no número de bits indicado em sua entrada a cada ciclo de relógio. Esta operação equivale a consumir bits da entrada de dados do parser, alinhando a posição atual no bitstream a fim de preparar os módulos decodificadores para identificar o próximo elemento sintático.

A escolha de um barrel-shifter e não um registrador de deslocamento comum se justifica pelos requisitos de desempenho do parser, que não pode esperar muitos ciclos para liberar a decodificação do próximo elemento sintático após uma operação.

5.6.3 Decodificador Exp-Golomb

O bloco responsável pela decodificação de elementos Exp-Golomb tem implementação completamente combinacional. Apesar de mais custosa em termos de uso de hardware quando comparada a uma implementação sequencial, novamente a escolha se justifica por requisitos de desempenho.

5.6.4 Decodificador de entropia CAVLC

O parser solicita a decodificação dos coeficientes para geração de resíduos para cada bloco de 4x4 pixels conforme a sintaxe do macrobloco que se está decodificando e o elemento *coded_block_pattern*, que indica quais partições de macrobloco (de dimensão 8x8) têm coeficientes não-nulos. Durante o processo, o CAVLD informa ao parser quantos bits

está consumindo da entrada a cada ciclo. Após a indicação de término do processo, o CAVLD atualiza sua memória de contexto adaptativo. No caso de blocos que não possuem resíduos – identificados pelo seu *coded_block_pattern* – o parser deve indicar ao CAVLD que não foram transmitidos coeficientes para estes blocos no bitstream, para que a memória de contexto possa ser atualizada.

O módulo de decodificação de entropia CAVLC tem complexidade elevada e sua arquitetura de implementação foge ao escopo deste trabalho. Para maiores detalhes, consultar [14]. A figura 20 apresenta um trecho de simulação, onde o módulo decodificador de slice data solicita ao CAVLD a decodificação dos coeficientes compactados. No decodificador de vídeo, a resposta fornecida é enviada através de uma FIFO aos módulos de quantização inversa e transformada inversa.

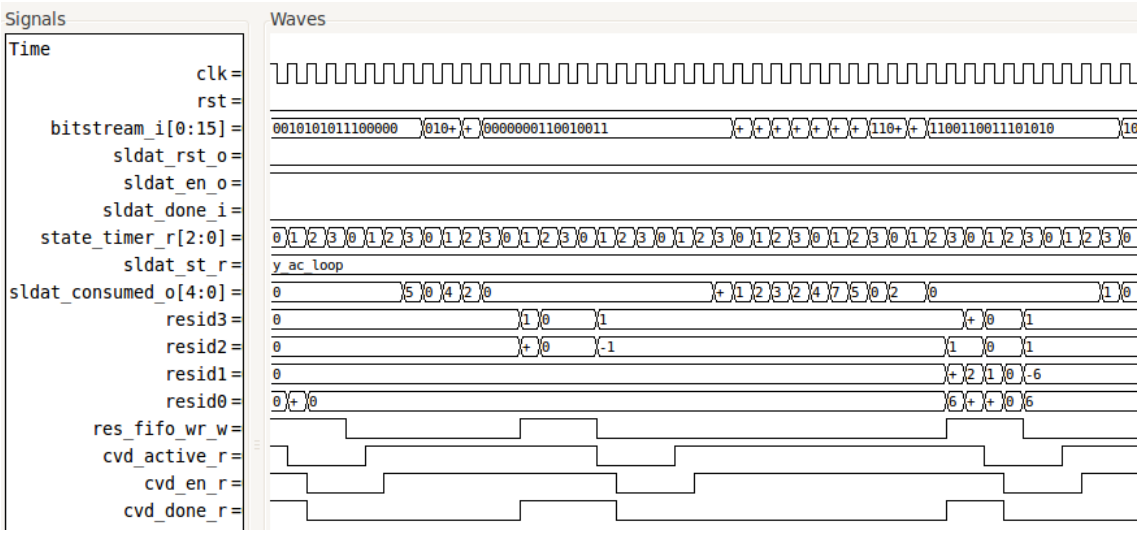

Figura 20: Solicitação de coeficientes ao CAVLD

6 RESULTADOS

6.1 Simulações

Conforme explicado no capítulo 4, as simulações são executadas sobre o sistema todo, e a validação é feita por comparação manual nos módulos decodificadores de SPS, PPS e slice header. Para o módulo decodificador de slice data, onde o volume de dados e elementos decodificados é bem maior, a comparação é feita automaticamente no testbench, comparando os coeficientes extraídos da decodificação de entropia para geração dos resíduos. Os resíduos corretos são retirados do PRH.264, como explicado na seção 4.3.3.

Dada a característica de codificar a grande maioria dos parâmetros do bitstream com tamanho variável de palavra, o método proposto de comparar apenas a geração de resíduos se mostra adequado. Supondo que haja um desalinhamento no consumo de bits da entrada devido a um erro na interpretação de algum elemento do bitstream, ao chegar na etapa de decodificação dos resíduos este desalinhamento de dados causa a busca por outra linha das tabelas da decodificação de entropia no CALVD, ocasionando a obtenção de resíduos que não correspondem aos esperados.

O parser desenvolvido foi simulado com diferentes bitstreams, tanto codificados com o codificador JM [6] quanto capturados das transmissões *one-seg* das emissoras locais. No total, mais de 100 quadros de vídeo com dimensões variadas foram simulados. O número de macroblocos simulados encontra-se na ordem de 30000. Para todos os casos, não houve erro na geração dos resíduos, indicando funcionamento adequado do parser.

6.2 Síntese para FPGA

A implementação do parser foi sintetizada para FPGA usando a ferramenta Xilinx ISE 10.1. Os dispositivos-alvo escolhidos foram o Xilinx Virtex II-Pro XC2VP30 e o Xilinx

Virtex-5 XC5VLX110T, existentes nas placas de desenvolvimento usadas no LaPSI. Os resultados de síntese do parser completo podem ser vistos na tabela 2.

Cabe observar que no Virtex II-Pro as LUTs são de 4 entradas, enquanto o Virtex-5 tem LUTs de 6 entradas. Além disso, a ferramenta de síntese não apresenta o número de slices para o Virtex-5.

Vale ressaltar que os resultados de síntese obtidos correspondem ao estado atual de desenvolvimento do decodificador de vídeo. À medida que novas funcionalidades forem adicionadas ao decodificador H.264 projetado no LaPSI, o decodificador pode requerer elementos sintáticos que no momento são identificados pelo parser, mas que por não terem saída declarada em seus módulos decodificadores, são eliminados no processo de síntese do mesmo. Isto faz com que, ao declarar uma saída a estes elementos já suportados, o registrador inferido para armazenamento do mesmo não mais será eliminado da síntese, aumentando levemente o uso de Flip-flops.

Tabela 2: Resultados de síntese do parser completo

	Virtex II-Pro XC2VP30			**Virtex-5 XC5VLX110T**		
Frequência	86.859MHz			127.257MHz		
Elemento	**Usado**	**Disponível**	**Utilização**	**Usado**	**Disponível**	**Utilização**
Slices	3137	13696	22%	—	—	—
Flip-flops	1198	27392	4%	1124	69120	1%
LUTs	6002	27392	21%	3541	69120	5%
Block RAMs	1	136	<1%	1	146	<1%

Para estimar o consumo de área de cada módulo interno ao parser, foram feitas sínteses parciais. Cada síntese envolve apenas um dos módulos projetados neste trabalho, além de alguns dos principais módulos herdados da implementação mínima anterior [13]. Os resultados podem ser vistos nas tabelas 3 a 9.

Tabela 3: Resultados de síntese do decodificador de SPS

	Virtex II-Pro XC2VP30			**Virtex-5 XC5VLX110T**		
Elemento	**Usado**	**Disponível**	**Utilização**	**Usado**	**Disponível**	**Utilização**
Slices	65	13696	<1%	—	—	—
Flip-flops	98	27392	<1%	71	69120	<1%
LUTs	69	27392	<1%	68	69120	<1%

Por estes resultados, pode-se observar que os módulos com maior consumo de recursos lógicos são o decodificador de entropia CAVLD, o barrel-shifter e o decodificador de slice data.

Tabela 4: Resultados de síntese do decodificador de PPS

	Virtex II-Pro XC2VP30			Virtex-5 XC5VLX110T		
Elemento	**Usado**	**Disponível**	**Utilização**	**Usado**	**Disponível**	**Utilização**
Slices	33	13696	<1%	—	—	—
Flip-flops	44	27392	<1%	41	69120	<1%
LUTs	43	27392	<1%	76	69120	<1%

Tabela 5: Resultados de síntese do decodificador de slice header

	Virtex II-Pro XC2VP30			Virtex-5 XC5VLX110T		
Elemento	**Usado**	**Disponível**	**Utilização**	**Usado**	**Disponível**	**Utilização**
Slices	62	13696	<1%	—	—	—
Flip-flops	76	27392	<1%	76	69120	<1%
LUTs	115	27392	<1%	104	69120	<1%

Tabela 6: Resultados de síntese do decodificador de slice data

	Virtex II-Pro XC2VP30			Virtex-5 XC5VLX110T		
Elemento	**Usado**	**Disponível**	**Utilização**	**Usado**	**Disponível**	**Utilização**
Slices	299	13696	2%	—	—	—
Flip-flops	170	27392	<1%	167	69120	<1%
LUTs	550	27392	2%	375	69120	<1%

Tabela 7: Resultados de síntese do decodificador de entropia CAVLD

	Virtex II-Pro XC2VP30			Virtex-5 XC5VLX110T		
Elemento	**Usado**	**Disponível**	**Utilização**	**Usado**	**Disponível**	**Utilização**
Slices	1231	13696	8%	—	—	—
Flip-flops	290	27392	1%	279	69120	<1%
LUTs	2285	27392	8%	1794	69120	2%

Tabela 8: Resultados de síntese do barrel-shifter

	Virtex II-Pro XC2VP30			Virtex-5 XC5VLX110T		
Elemento	**Usado**	**Disponível**	**Utilização**	**Usado**	**Disponível**	**Utilização**
Slices	763	13696	5%	—	—	—
Flip-flops	74	27392	<1%	70	69120	<1%
LUTs	1490	27392	5%	766	69120	1%

Tabela 9: Resultados de síntese do decodificador de Exp-Golomb

	Virtex II-Pro XC2VP30			Virtex-5 XC5VLX110T		
Elemento	**Usado**	**Disponível**	**Utilização**	**Usado**	**Disponível**	**Utilização**
Slices	116	13696	<1%	—	—	—
LUTs	196	27392	<1%	139	69120	<1%

7 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um módulo de hardware descrito em VHDL para implementação de um parser para um decodificador de vídeo H.264/AVC. A solução apresentada é capaz de interpretar todos os elementos sintáticos presentes nas transmissões *one-seg* do SBTVD, com exceção do conjunto de parâmetros VUI. Além disso, a solução já fornece os dados necessários para a predição inter-quadros das transmissões *full-seg*, codificadas sob o perfil High do padrão H.264, ou seja, provê os vetores de movimento e referências na lista de imagens para os métodos de bipredição presentes neste perfil.

A descrição da arquitetura foi feita de modo a facilitar a expansão das funcionalidades, prevendo estados para interação com máquinas de estado menores comandadas por uma máquina hierarquicamente superior.

O módulo já está sendo utilizado pelos membros responsáveis pela integração global do decodificador de vídeo. Elementos que no princípio eram configurados manualmente antes da síntese na descrição VHDL agora são extraídos do bitstream, facilitando a tarefa dos desenvolvedores e poupando o tempo de ressintetizar o projeto a cada vez que as dimensões do vídeo ou o fator de quantização dos resíduos era alterado de um bitstream para outro nos testes.

O desenvolvimento deste trabalho também contribui para a tarefa de integração do módulo de compensação de movimento utilizado na predição inter-quadros. O parser alimenta uma FIFO com os parâmetros necessários para o funcionamento deste módulo, tornando mais transparente a interface do mesmo com o parser.

Como trabalhos futuros, deve-se implementar as máquinas de estado responsáveis pela interpretação dos elementos avançados dos pacotes SPS, PPS e do slice header. Deve-se também implementar módulos para gerenciamento de múltiplos conjuntos de

parâmetros, possivelmente armazenando em memória externa os conjuntos de parâmetros que não estão ativos na decodificação da imagem atual. Após a conclusão do desenvolvimento do módulo CABAD, o mesmo deve ser integrado a este parser.

No âmbito pessoal, este trabalho me possibilitou a experiência de desenvolver um componente de um projeto maior, desenvolvendo habilidades de trabalho em equipe, busca de informações, documentação e apresentação de resultados. Contribuiu também largamente com a prática de projeto de hardware em seus mais variados aspectos, como controle de módulos variados, uso de FIFOs, balanceamento entre desempenho e consumo de elementos lógicos, simulações comportamentais, entre outros.

Além disso, tanto este trabalho quanto o período anterior de trabalho como bolsista do LaPSI me proporcionaram contato maior com o meio acadêmico através da escrita e apresentação de artigos, participação em eventos da comunidade científica. A experiência de estar engajado num grupo de pesquisa com profissionais muito bem qualificados complementou a formação curricular do curso de graduação.

REFERÊNCIAS

[1] I. E. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for the Next-generation Multimedia*. John Wiley and Sons, 2003.

[2] F. SBTVD, "Fórum do sistema brasileiro de TV digital terrestre." http://www.forumsbtvd.org.br/, Maio 2010.

[3] R. H. SBTVD, "Wiki rede H.264 SBTVD." http://www.lapsi.eletro.ufrgs.br/h264/wiki, Julho 2010.

[4] ITU-T, *ITU-T Recommendation H.264 (03/05): Advanced video coding for generic audiovisual services*. International Telecommunication Union, 2005.

[5] ABNT, *NBR15602 Televisão Digital Terrestre - Codificação de vídeo, áudio e multiplexação*. ABNT, 2007.

[6] H. S. Coordination, "JM software." http://iphome.hhi.de/suehring/tml/, Maio 2010.

[7] V4L, "Video4linux." http://linux.bytesex.org/v4l2/, Julho 2010.

[8] LinuxTV, "Linux TV DVB apps." http://www.linuxtv.org/wiki/index.php/LinuxTV_dvb-apps, Julho 2010.

[9] M. A. Lorencetti, W. T. Staehler, and A. A. Susin, "Reference C software H.264/AVC decoder for hardware debug and validation," in *XXIII South Symposium on Microelectronics* (SBC, ed.), (Bento Gonçalves-RS), pp. 127–130, 2008.

[10] M. A. Lorencetti, W. T. Staehler, and A. A. Susin, "Incremental hardware development from modular mixed C-VHDL simulation," in *8th Students Forum on Microelectronics* (SBC, ed.), (Gramado-RS), 2008.

[11] wxWidgets, “Cross-platform GUI library.” http://www.wxwidgets.org/, Julho 2010.

[12] SDL, “Simple directmedia layer.” http://www.libsdl.org/, Julho 2010.

[13] F. I. Pereira, “Hardware implementation of a high performance minimalist H.264 video decoder,” Master’s thesis, Helsinki Metropolia University of Applied Sciences, 2009.

[14] T. Silva, F. Pereira, A. Susin, S. Bampi, and L. Agostini, “High performance and low cost architecture for H.264/AVC CAVLD targeting HDTV,” in *SBCCI ’09: Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design*, (Natal, Brazil), pp. 1–5, ACM, 2009.